EXHIBIT 13

8464067

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

March 6, 2024

THIS IS TO CERTIFY THAT ANNEXED IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF THE FILE WRAPPER AND CONTENTS OF:

APPLICATION NUMBER: *10/939,903*
FILING DATE: *September 13, 2004*
PATENT NUMBER: *7,519,814*
ISSUE DATE: *April 14, 2009*

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office

Breaun Johnson
Certifying Officer

15750 U.S. PTO

Case No. 78802 (120-1 US)
091304

COMMISSIONER FOR PATENTS
PO BOX 1450
ARLINGTON, VA 22313-1450

I HEREBY CERTIFY THIS PAPER OR FEE IS BEING DEPOSITED WITH THE U.S. POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 CFR 1.10 ON THE DATE INDICATED BELOW AND IS ADDRESSED TO: COMMISSIONER FOR PATENTS, PO BOX 1450, ALEXANDRIA, VA 22313-1450.

EXPRESS MAIL NO.: EL 964278525 US

DATE OF DEPOSIT: September 13, 2004

NAME: Kristen Ferguson

SIGNATURE: Kristen Ferguson

22264 U.S. PTO
10/939903
091304

Transmitted herewith for filing is the patent application of:

Inventors: Donn ROCHETTE, Paul O'LEARY, Dean HUFFMAN

For: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

Please note the following:

[X] Patent Application: 37 pages, 34 claims.
[X] 17 sheets of drawings.
[X] The suggested drawing figure to be published is FIG. 1
[X] A declaration and power of attorney.
[ ] Citation Under 37 CFR 1.97 and PTO-1449.
[X] Assignee info:
Name: Trigence Corp.
Address: 750 Palladium Drive, Ste. 210
Ottawa, Ontario, CANADA K2V 1C7
State of Incorporation:
[X] Applicant qualifies as a small entity under 37 CFR ' 1.27.

[X] Applicant claims priority benefit to the following U.S. application(s):
Application No.: 60/502,619
Filing Date: SEPTEMBER 15, 2003
Application No.: 60/512,103
Filing Date: OCTOBER 20, 2003

The filing fee has been calculated as shown below:

	(Col. 1)		(Col. 2)		SMALL ENTITY			LARGE ENTITY	
FOR:	# FILED		# EXTRA		RATE	FEE		RATE	FEE
BASIC FEE						$ 385	OR		$
TOTAL CLAIMS	34	-20	14		X 9	$ 126	OR	X 18	$
INDEP CLAIMS	2	- 3			X 43	$	OR	X 86	$
* If the difference in Col. 1 is less than "0", enter "0" in Col. 2.					TOTAL	$ 511		TOTAL	$

[X] Authorization is given to charge the Filing Fee in the amount of $511.00 to Deposit Account 50-2810.

[X] Authorization is given to charge the Assignment Recordal Fee in the amount of $40.00 to Deposit Account 50-2810.

[X] The Commissioner is authorized to charge or credit any discrepancies in fee amounts to Deposit Account No. 50-2810.

[X] PLEASE ADDRESS ALL CORRESPONDENCE TO ATTORNEY OF RECORD: CHARLES E. WANDS

[X] **Please associate this application with Customer No. 27975.**

CUSTOMER NO. 27975

Telephone: (321) 725-4760

September 13, 2004

CHARLES E. WANDS
Reg. No. 25,649

15750 U.S. PTO

Case No. 78802 (120-1 US) 091304

COMMISSIONER FOR PATENTS
PO BOX 1450
ARLINGTON, VA 22313-1450

I HEREBY CERTIFY THIS PAPER OR FEE IS BEING DEPOSITED WITH THE U.S. POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 CFR 1.10 ON THE DATE INDICATED BELOW AND IS ADDRESSED TO: COMMISSIONER FOR PATENTS, PO BOX 1450, ALEXANDRIA, VA 22313-1450.

EXPRESS MAIL NO.: EL 964278525 US

DATE OF DEPOSIT: September 13, 2004

NAME: Kristen Ferguson

SIGNATURE: Kristen Ferguson

22264 U.S. PTO 10/939903 091304

Transmitted herewith for filing is the patent application of:

Inventors: Donn ROCHETTE, Paul O'LEARY, Dean HUFFMAN

For: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

Please note the following:

[X] Patent Application: 37 pages, 34 claims.
[X] 17 sheets of drawings.
[X] The suggested drawing figure to be published is FIG. 1
[X] A declaration and power of attorney.
[ ] Citation Under 37 CFR 1.97 and PTO-1449.
[X] Assignee info:
Name: Trigence Corp.
Address: 750 Palladium Drive, Ste. 210
Ottawa, Ontario, CANADA K2V 1C7
State of Incorporation:
[X] Applicant qualifies as a small entity under 37 CFR ' 1.27.

[X] Applicant claims priority benefit to the following U.S. application(s):
Application No.: 60/502,619
Filing Date: SEPTEMBER 15, 2003
Application No.: 60/512,103
Filing Date: OCTOBER 20, 2003

The filing fee has been calculated as shown below:

	(Col. 1)		(Col. 2)		SMALL ENTITY			LARGE ENTITY	
FOR:	# FILED		# EXTRA		RATE	FEE		RATE	FEE
BASIC FEE						$ 385	OR		$
TOTAL CLAIMS	34	-20	14		X 9	$ 126	OR	X 18	$
INDEP CLAIMS	2	- 3			X 43	$	OR	X 86	$
* If the difference in Col. 1 is less than "0", enter "0" in Col. 2.					TOTAL	$ 511		TOTAL	$

[X] Authorization is given to charge the Filing Fee in the amount of $511.00 to Deposit Account 50-2810.

[X] Authorization is given to charge the Assignment Recordal Fee in the amount of $40.00 to Deposit Account 50-2810.

[X] The Commissioner is authorized to charge or credit any discrepancies in fee amounts to Deposit Account No. 50-2810.

[X] PLEASE ADDRESS ALL CORRESPONDENCE TO ATTORNEY OF RECORD: CHARLES E. WANDS

[X] **Please associate this application with Customer No. 27975.**

CUSTOMER NO. 27975

Telephone: (321) 725-4760

September 13, 2004

CHARLES E. WANDS
Reg. No. 25,649

DOC. NO. 78802 120-1 US

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

Cross-Reference to Related Applications

[1] This application claims priority of U.S. Provisional Patent Application Nos: 60/502,619 filed September 15, 2003 and 60/512,103 filed October 20, 2003, which are incorporated herein by reference for all purposes.

Field of the Invention

[2] The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

Background of the Invention

[3] Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

[4] The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

[5] In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

[6] In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

[7] As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

[8] There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

[9] A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

[10] This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

[11] The following definitions are used herein:

[12] •Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

[13] •Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

[14] •Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

[15] •Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

[16] •Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

[17] • System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

[18] By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

[19] /usr/lib/libz.so.1

[20] /lib/libssl.so.2

[21] /lib/libcrypto.so.2

[22] /usr/lib/libaprutil.so.0

[23] /usr/lib/libgdbm.so.2

[24] /lib/libdb-4.0.so

[25] /usr/lib/libexpat.so.0

[26] /usr/lib/libapr.so.0

[27] /lib/i686/libm.so.6

[28] /lib/libcrypt.so.1

[29] /lib/libnsl.so.1

[30] /lib/libdl.so.2

[31] /lib/i686/libpthread.so.0

[32] /lib/i686/libc.so.6

[33] /lib/ld-linux.so.2

[34] Apache uses the following configuration files, also provided with the OS distribution:

[35] /etc/hosts

[36] /etc/httpd/conf

[37] /etc/httpd/conf.d

[38] /etc/httpd/logs

[39] /etc/httpd/modules

[40] /etc/httpd/run

[41] By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files

included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

Summary of the Invention

[42] In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

[43] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

[44] In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

[45] a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more

processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

[46] In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

[47] Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

[48] Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

[49] In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

[50] In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySql and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

[51] In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

[52] In an embodiment of the invention, each respective secure application container has data files for the at least one application within the respective secure application container.

[53] The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

[54] In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

[55] In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

[56] Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

[57] In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

[58] In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

[59] For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

[60] By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

[61] If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

[62] Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

[63] If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

[64] In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the

server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

[65] In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

[66] More particularly, the method also involves, during execution on the computer system:

[67] monitoring resource usage of the at least one respective application associated with the group;

[68] intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

[69] comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

[70] and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

[71] The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

[72] Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of this invention may choose to return a

container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed "spoofing" and will be explained in greater detail within the disclosure.

Brief Description of the Drawings

[73] Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

[74] Figure 1 is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

[75] Figure 2 is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

[76] Figure 3 is a pictorial diagram illustrating the creation of a container.

[77] Figure 4 is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the "kernel module" which is used to handle system calls.

[78] Figure 5 is a diagram similar to Figure 4, wherein additional configuration data is provided.

[79] Figure 6 is a diagram illustrating a system wherein the containers are secure containers.

[80] Figure 7 is a diagram introducing the sequencing or order in which applications within a container are executed.

[81] Figure 8 is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

[82] Figure 9 is a diagram that illustrates the installation of a container on a server.

[83] Figure 10 is a diagram that illustrates the monitoring of a number of applications and state information.

[84] Figure 11 is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

[85] Figure 12 is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

[86] Figure 13 is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

[87] Figure 14 is a screen snapshot of an implementation according to the schematic of Figure 13.

[88] Figure 15 is a flow chart of a method of initializing icons of Figure 14.

[89] Figure 16 is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of Figure 14; and,

[90] Figure 17 is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of Figure 13.

Detailed Description

[91] Turning now to Figure 1, a system is shown having a plurality of servers 10a, 10b. Each server includes a processor and an independent operating system. Each operating system includes a kernel 12, hardware 13 in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs 11a, 11b and related system files 11c. These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

[92] Figure 2 illustrates a system in accordance with the invention in which multiple applications 21a, 21b, 21c, 21d, 21e, and 21f can execute in a secure environment on a single server. This is made possible by associating applications with secure containers 20a, 20b and 20c. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

[93] Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

[94] The Secure application containers 20a, 20b and 20c are shown in Figure 2 in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

[95] The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

[96] Referring now to Figure 3 the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform 32 where an application of interest is currently installed or the files can be extracted from install media or package files 30. Two methods of container creation will be described hereafter.

[97] Figure 4 illustrates a system having two secure containers 20a 20b, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform 40 hosting containers. One embodiment shows these services implemented in a kernel module 43. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container 20a, 20b environment system calls are intercepted, by the kernel module 43 passing through services installed as part of this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by “spoofing” the system.

[98] As introduced heretofore, "spoofing" is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system's kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the 'uname' system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has 'spoofed' the 'uname' system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same "spoofing" mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

[99] By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

[100] System calls are intercepted to perform the following services:

[101] tracking applications, for example a tracking of which applications are in and out of a specific container;

[102] spoofing particular values returned by the kernel;

[103] applying resource checks and imposing resource limits;

[104] monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,

[105] retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used

and information related to socket usage, such as the amount of data transferred through a given socket connection.

[106] Container configuration includes the definition of hardware resource usage, as depicted in Figure 5 including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container 20a or 20b. These values can be used to determine resource requirements for a given compute platform 40. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user 51 when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

[107] It can be seen from Figure 6 that each application executing associated with a container 20a or 20b, or contained within a container 20a or 20b, is able to accesses files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform 40 upon which they execute. Moreover, applications with a container 20a association are not able to access the files contained in another container 20b.

[108] When a container 20a or 20b is installed specific applications are started, as is shown in Figure 7 and container configuration information defines how applications 71, 72 are started. This includes the definition of a first program to start when a container is installed on a compute platform 40. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script 73, delineating the first

application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

[109] Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module 43 in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

[110] The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in Figure 7, that the current process is part of the container. Then, the application 71 is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

[111] A container has a physical presence when not installed on a compute platform 40. It is described as residing on a network accessible repository. As is shown in Figure 8, a container has a physical presence in two locations 82, 84; or stated differently, the files associated with a container have a physical presence in two locations 82, 84. The files used by applications associated with a container reside on a network file server 82. Container configuration is managed by a controller. The configuration state is organized in a database 84 used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

[112] In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

[113] When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. Figure 9 shows that the files 82a, 84a are either copied, physically placed on storage local to the compute platform 40, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

[114] Configuration information is used to direct how the container is installed. This includes operations such as describing the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

[115] Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module 43 in Figure 9.

[116] As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. Figure 10 shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

[117] Figure 3 illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

[118] These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform.

Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform an then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

[119] Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

[120] Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

[121] Referring once again to Figure 3, applications may come from third party installation media on CDROM, package files 30, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

[122] The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in Figure 3. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server 32, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

[123] The description of the two methods above differs in that in one instance the container creation begins with the system files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

[124] Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

[125] modifying the contents a file to add a container specific IP address;

[126] modifying the contents of a directory to define start scripts that should be executed;

[127] modifying the contents of a file to define which mount points will relate to a container;

[128] removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

[129] In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

[130] This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

[131] In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

[132] In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

[133] A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

[134] Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

[135] This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

[136] Drag and Drop Application Management

[137] Another aspect of this relates to software that manages the operation of a data center.

[138] There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

[139] An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

[140] In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

[141] The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

[142] In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

[143] Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

[144] In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

[145] In some embodiments, upon initialization, a user account is created for the selected software application.

[146] In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

[147] In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

[148] In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

[149] In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

[150] The computing platform may be a remote computing platform.

[151] In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

[152] In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

[153] In some embodiments, the pointing device is a mouse.

[154] In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

[155] In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

[156] In some embodiments, upon initialization, any process associated with the selected software application is stopped.

[157] In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

[158] In some embodiments, upon initialization, a user account associated with the selected software application is removed.

[159] In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

[160] The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

[161] The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

[162] The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

[163] The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

[164] Selection can be made using a drag and drop operation.

[165] The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

[166] Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to

select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

[167] The following definitions are used herein:

[168] Storage repository: A centralized disk based storage containing application specific files that make up a software application.

[169] GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

[170] Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

[171] Aspects of the invention include:

[172] •GUI supporting drag and drop operations

[173] •Icons

[174] •Storage medium

[175] •Console

[176] •Network connections

[177] •One or more computing platforms

[178] While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

[179] When a software application icon is "dropped" on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

[180] Referring to Figure 12, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

[181] A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

[182] Referring to Figure 13, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium, as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

[183] Referring to Figure 14, a screen snapshot of an implementation is shown according to the schematic of Figure 13. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

[184] Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in Figure 14, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

[185] Referring to Figure 15, shown is a flow chart of a method of initializing the icons of Figure 14. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

[186] With reference to Figures 12 to 14, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

[187] •Tests whether the computing platform is a valid computing platform;

[188] •Creates a user account for the software application;

[189] •Installs application specific files so that they are accessible to the remote computing platform;

[190] •Sets file access permissions such that a new user can access its applications;

[191] •Starts a first application; and

[192] •Updates a console showing that the selected software application is resident on the selected computing platform.

[193] These steps will now be described with reference to Figure 16 which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of Figure 2. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

[194] The method steps of Figure 16 are performed by an install drop handler. During installation, verification is performed to determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

[195] A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

[196] With reference to Figure 17, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in Figure 15), each application software installed on the computing platform.

[197] The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

[198] •Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.

[199] •Stops processes associated with the software application;
[200] •Copies application specific data file changes from the computing platform back to the storage medium;
[201] •Removes user accounts associated with the software application; and
[202] •Updates the console to show that the selected software application is resident in the repository.

[203] These steps will now be described with reference to Figure 17 which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of Figure 13. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the

storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

[204] Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of Figures 16 and 17, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

[205] A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

[206] Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

Claims

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

4. A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim 2, further comprising the step of modifying at least some of the system files to provide an association with a container specific identity assigned to the container.

6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim 2, wherein in operation when applications are executed, applications within a container have no access to system files or applications in other containers or system files within the operating system during execution thereof if those applications or system files are read/write files.

11. A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.

13. A method as defined in claim 1 further comprising the step of associating with a container a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:
a) running an instance of a service on a server;
b) determining which files are being used; and,
c) copying applications and associated system files to memory.

15. A method as defined in claim 14 comprising the steps of:
assigning an identity to the containers including at least one of a unique IP address, a unique MAC address and an estimated resource allocation;
installing the container on a server; and,
testing the applications and files within the container.

16. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:
using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one

or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

18. A computing system as defined in claim 17, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

19. A computing system as defined in claim 18, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the MAC_ Address.

20. A computing system as defined in claim 17, wherein the run time module performs:

monitoring resource usage of applications executing;

intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and,

forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

21. A method as defined in claim 1 further comprising the step of installing a service on a target server selected from one of the plurality of servers, wherein the step of installing the service includes the steps of:

using a graphical user interface, associating a unique icon representing a service with an unique icon representing a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

22. A method as defined in claim 21 wherein the target server and the graphical user interface are at remote locations.

23. A method as defined in claim 22, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

24. A method as defined in claim 23, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

25. A method claim 21, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

26. A method according to claim 21 further comprising, the step of creating a user account for the service.

27. A method as defined in claim 21, further comprising the step of installing files specific to the selected application on the selected server.

28. A method according to claim 21 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

29. A method as defined in claim 24 further comprising the step of starting a distributed software application.

30. A method according to anyone of claims 24 further comprising the steps of updating a console on the selected target server with information indicating that the service is resident on the selected target server.

31. A method as defined in claim 1, further comprising the step of: de-installing a service from a server, comprising the steps of:
displaying the icon representing the service;
displaying a the icon representing the server on which the service is installed;
and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

32. A method according to claim 31 further comprising the step of separating icon representing the service from the icon representing the server.

33. A method according to claim 31 further comprising the step of testing whether the selected server is a valid computing platform for de-installation of the service.

34. A method according to claim 31 further comprising the step of copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

Abstract of the Disclosure

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

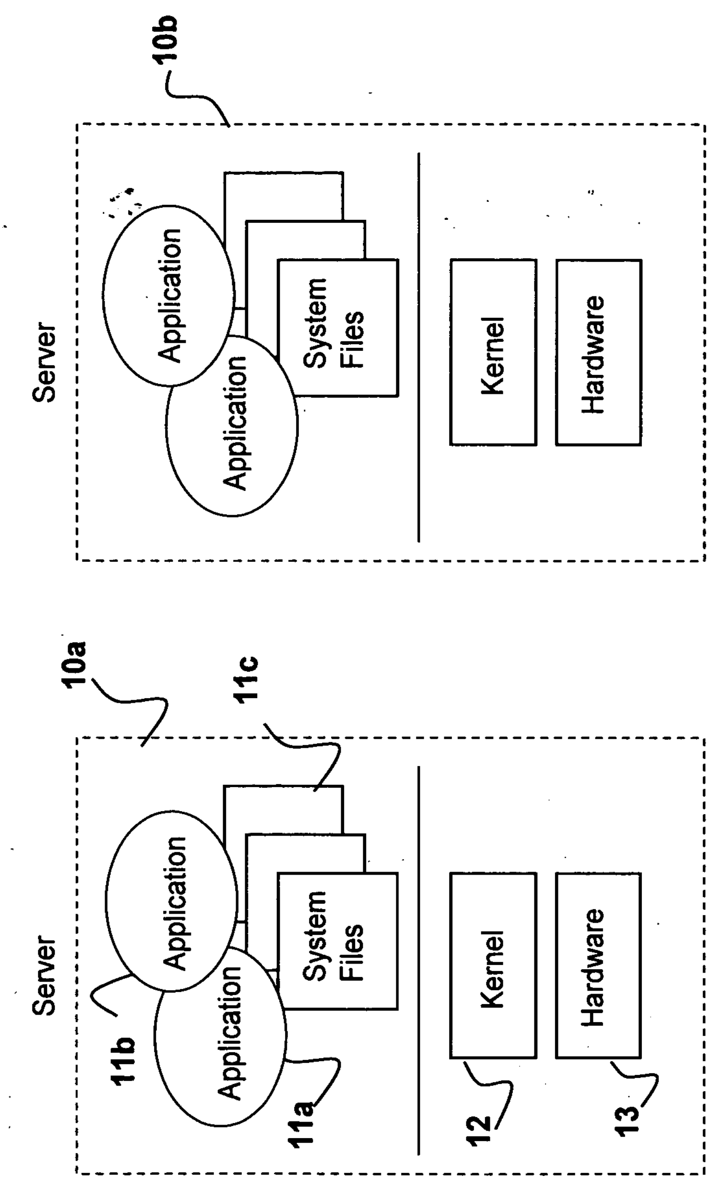

10a
10b
Server
Application
Application
System Files
Kernel
Hardware
11a
11b
11c
12
13
Server
Application
Application
System Files
Kernel
Hardware

Figure 1

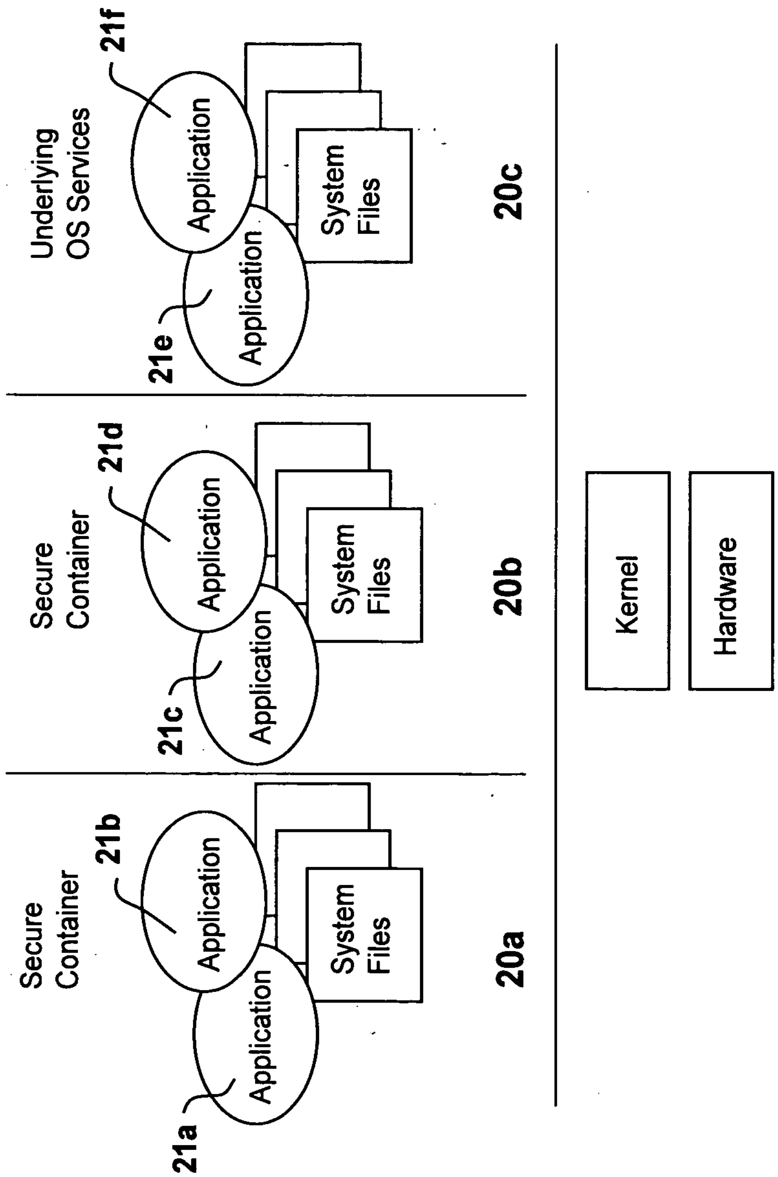
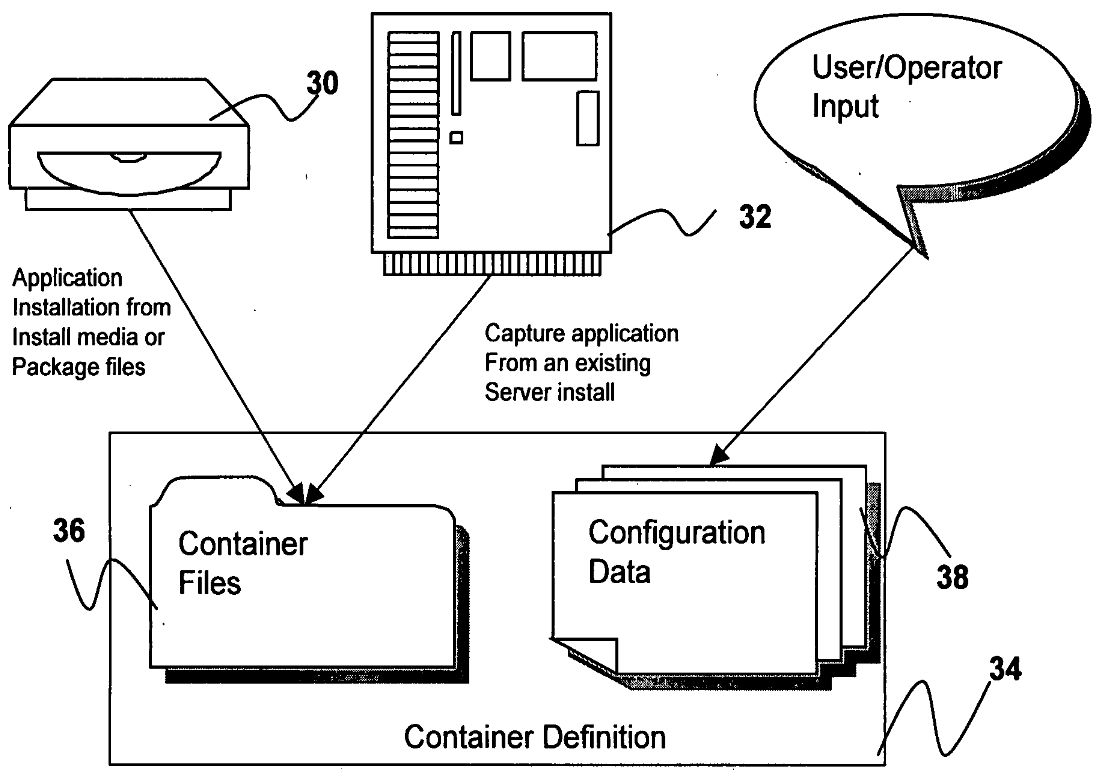

Secure Container
Secure Container
Underlying OS Services
Application
Application
Application
Application
Application
Application
System Files
System Files
System Files
21a
21b
21c
21d
21e
21f
20a
20b
20c
Kernel
Hardware

Figure 2



30
32
User/Operator
Input
Application
Installation from
Install media or
Package files
Capture application
From an existing
Server install
36
Container
Files
Configuration
Data
38
34
Container Definition

Figure 3

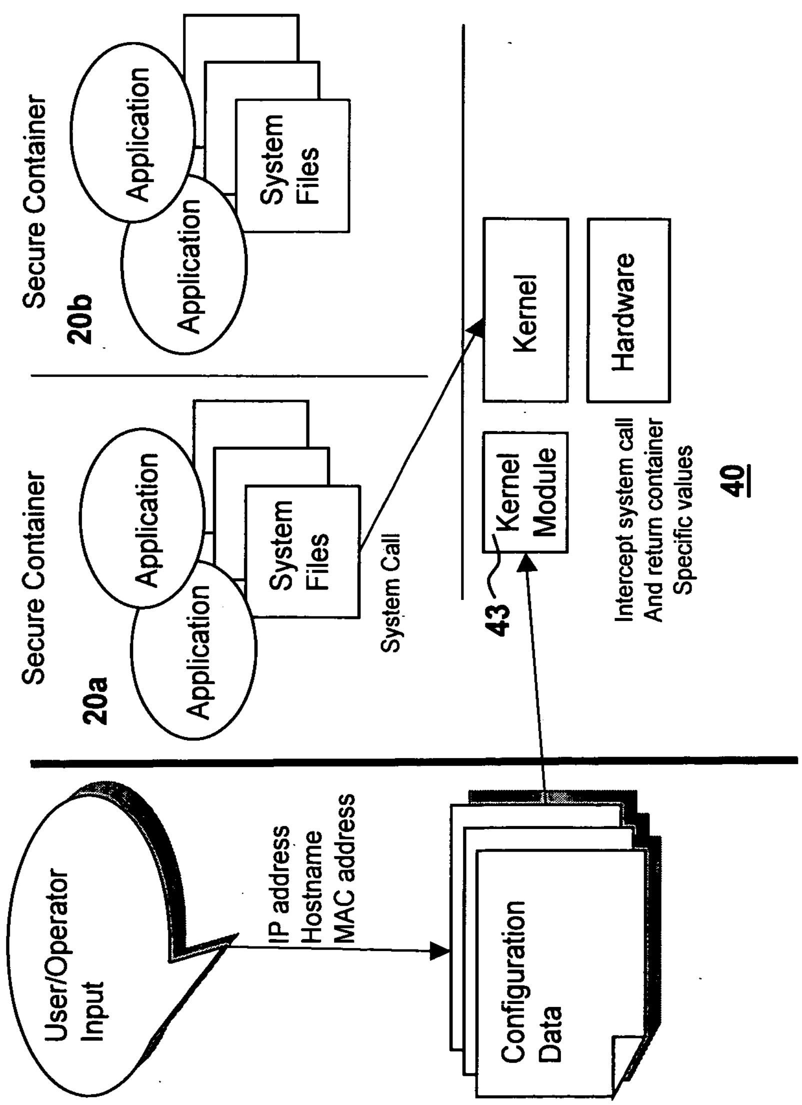
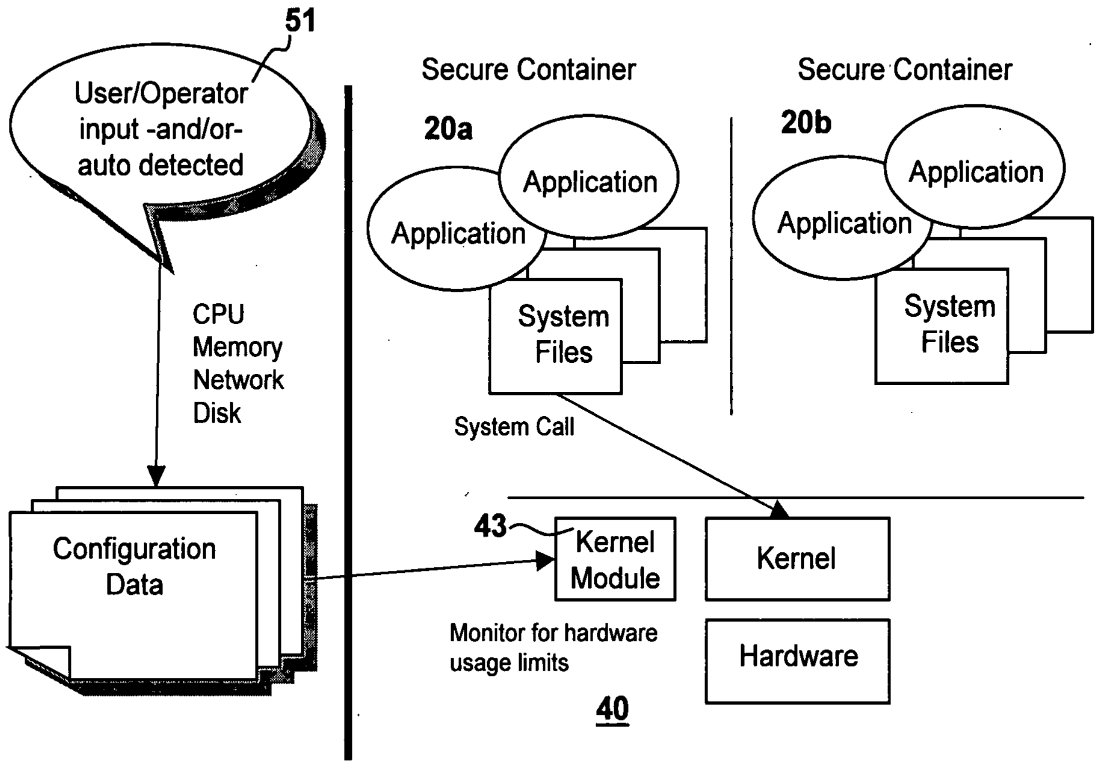
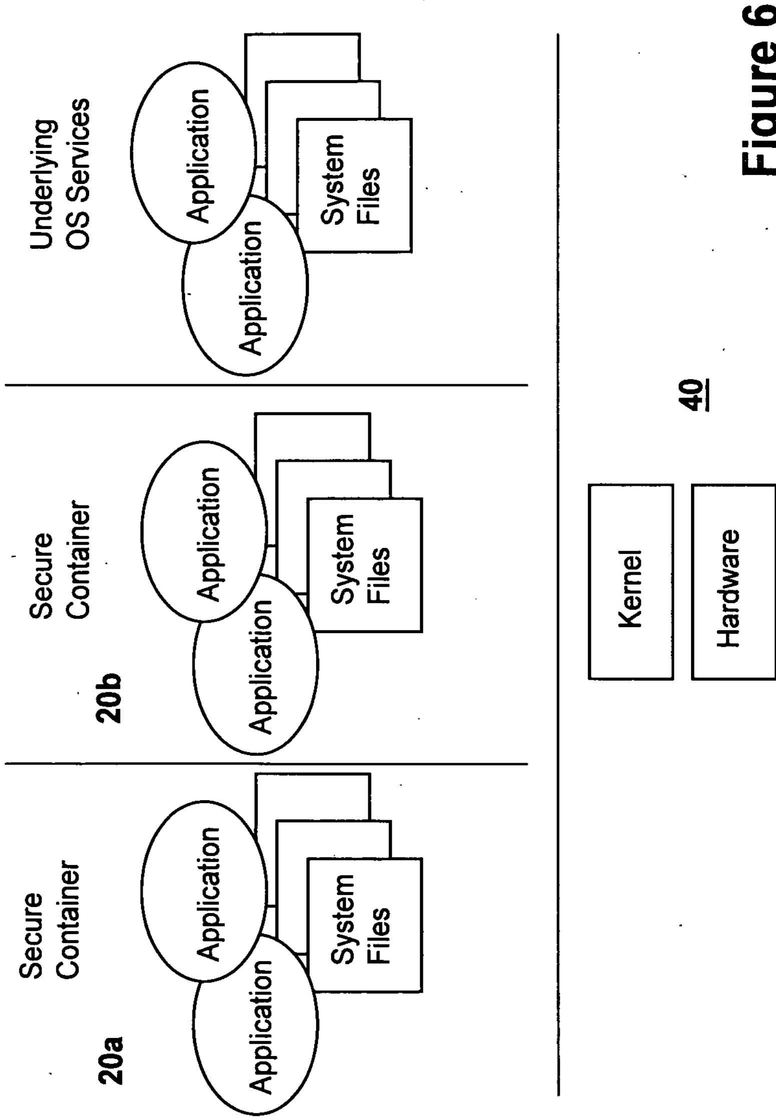

User/Operator Input
IP address
Hostname
MAC address
Configuration Data
Secure Container
20a
Application
Application
System Files
System Call
Secure Container
20b
Application
Application
System Files
Kernel
Hardware
Kernel Module
43
Intercept system call
And return container
Specific values
40

Figure 4



51
User/Operator input -and/or- auto detected
CPU
Memory
Network
Disk
Configuration Data
Secure Container
20a
Application
Application
System Files
Secure Container
20b
Application
Application
System Files
System Call
43
Kernel Module
Kernel
Hardware
Monitor for hardware usage limits
40

Figure 5



20a
Secure Container
Application
Application
System Files
20b
Secure Container
Application
Application
System Files
Underlying OS Services
Application
Application
System Files
Kernel
Hardware
40

Figure 6

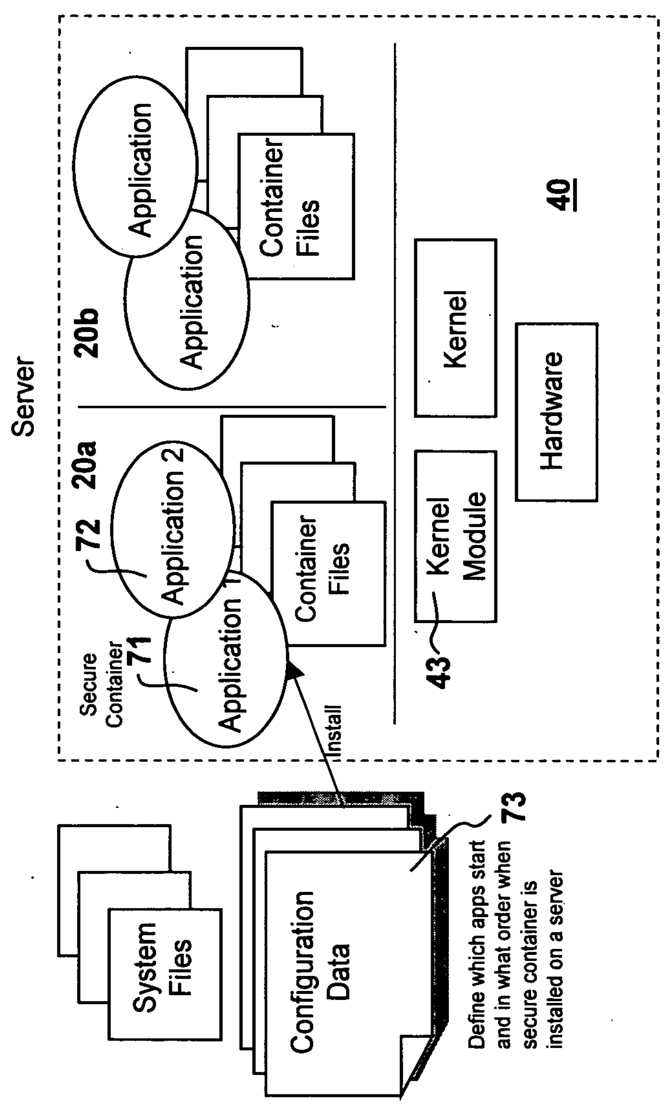
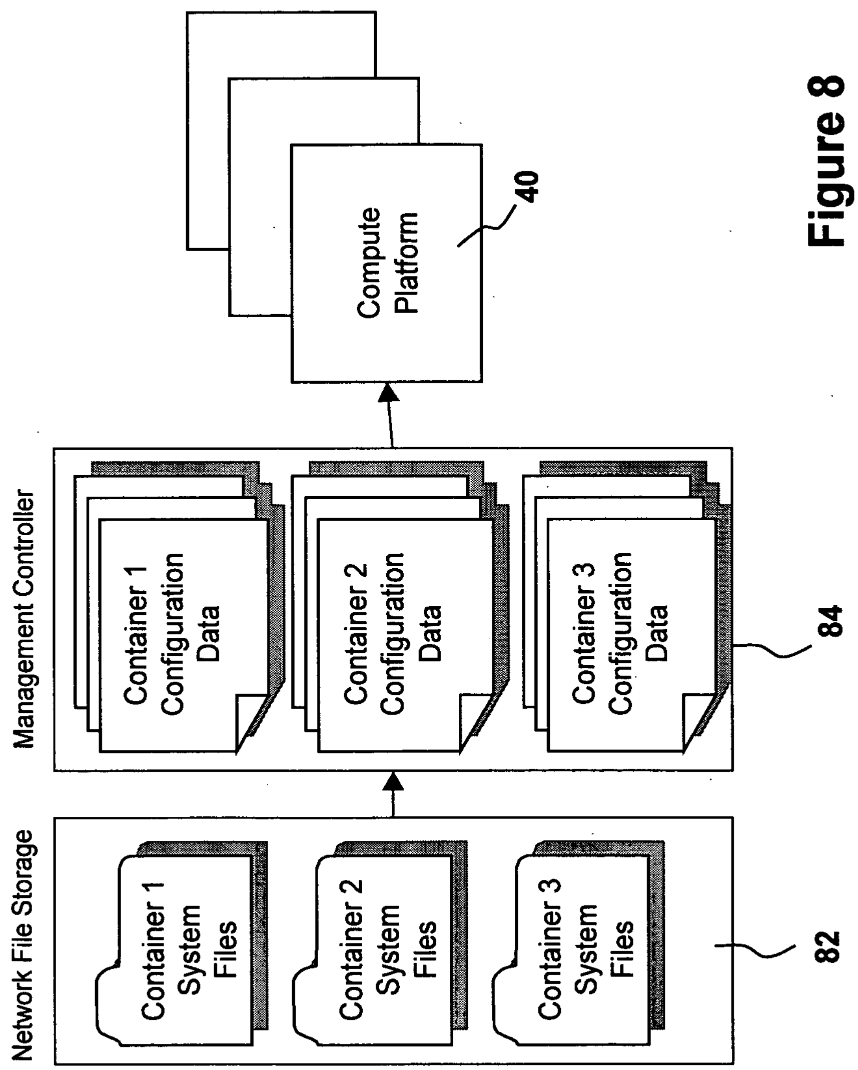

Server
Secure Container
71
72
Application 1
Application 2
20a
20b
Application
Application
Container Files
Container Files
Kernel Module
43
Kernel
Hardware
40
System Files
Configuration Data
Install
73
Define which apps start and in what order when secure container is installed on a server

Figure 7



Network File Storage
Container 1 System Files
Container 2 System Files
Container 3 System Files
82
Management Controller
Container 1 Configuration Data
Container 2 Configuration Data
Container 3 Configuration Data
84
Compute Platform
40

Figure 8

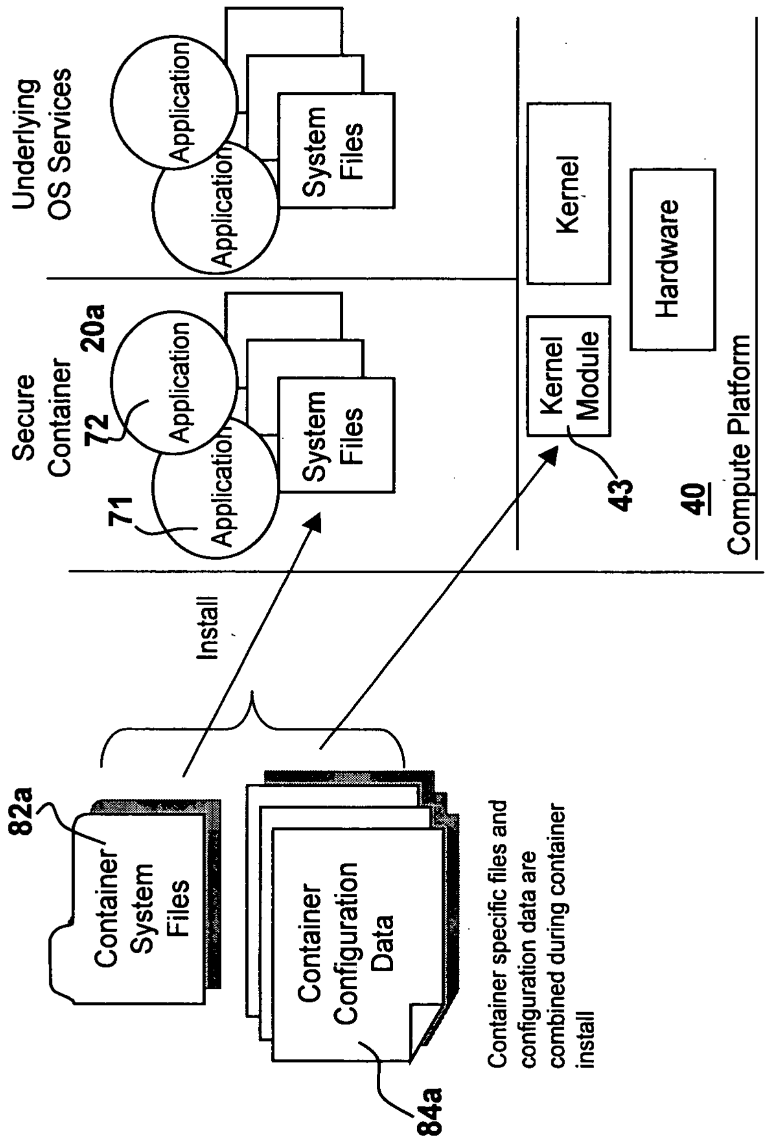

Underlying
OS Services
Application
Application
System
Files
Secure
Container
20a
72
71
Application
Application
System
Files
Kernel
Kernel
Module
Hardware
43
40
Compute Platform
Install
82a
Container
System
Files
Container
Configuration
Data
84a
Container specific files and configuration data are combined during container install

Figure 9

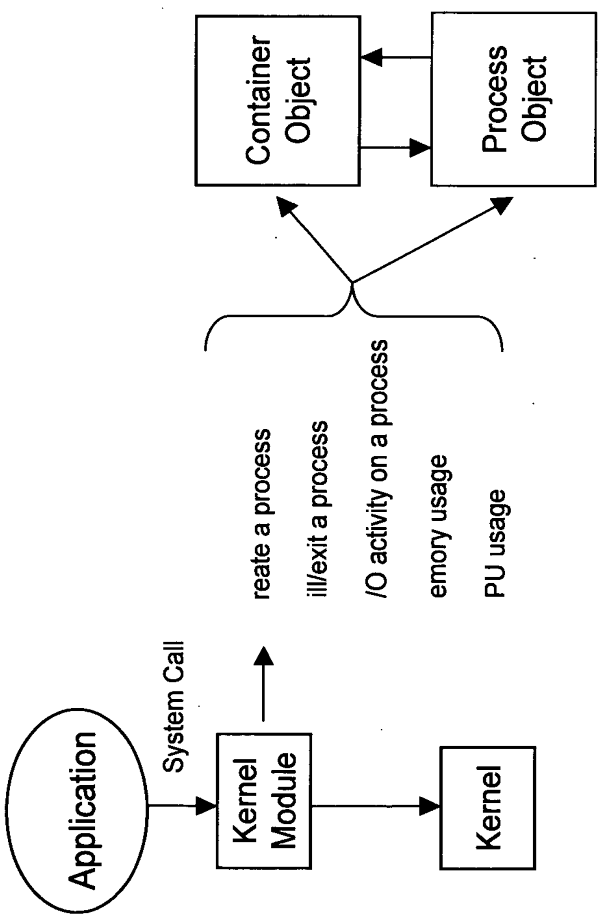

Application
System Call
Kernel Module
Kernel
reate a process
ill/exit a process
/O activity on a process
emory usage
PU usage
Container Object
Process Object

Figure 10

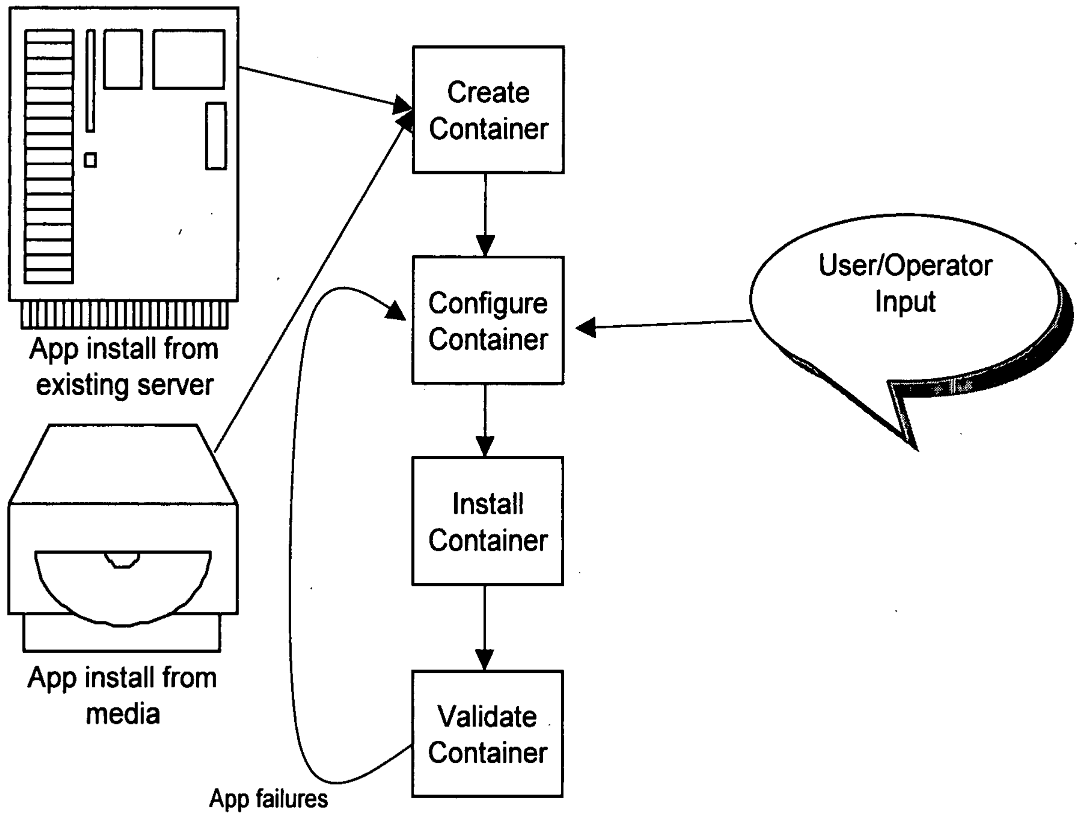

App install from existing server
App install from media
Create Container
Configure Container
Install Container
Validate Container
App failures
User/Operator Input

Figure 11

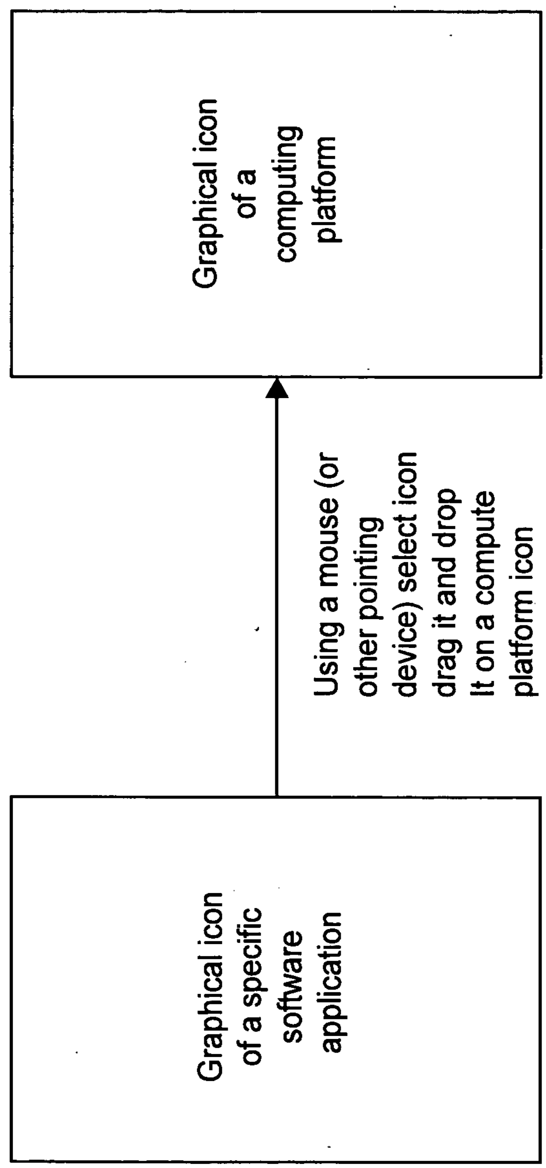

Graphical icon of a specific software application
Using a mouse (or other pointing device) select icon drag it and drop It on a compute platform icon
Graphical icon of a computing platform

Figure 12

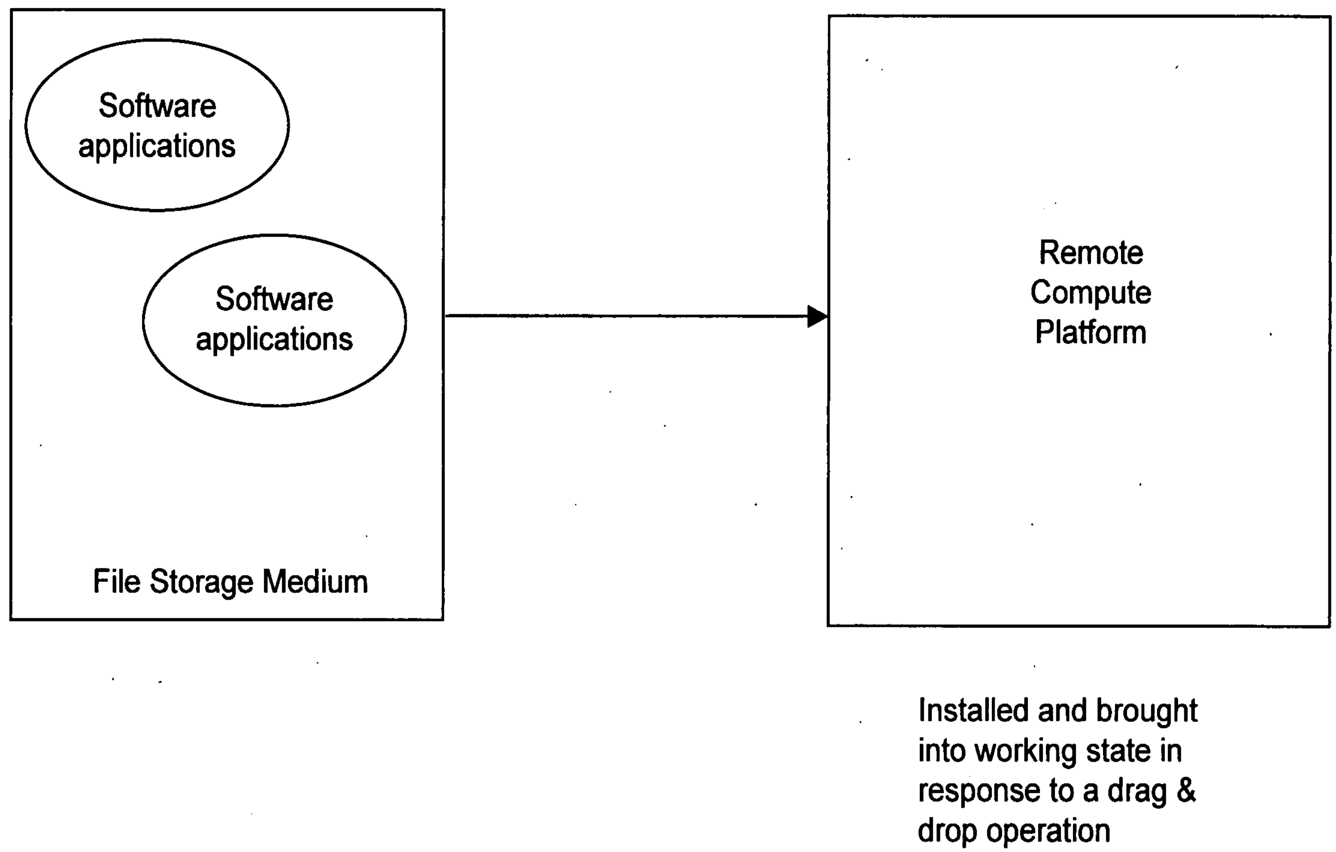

Software applications
Software applications
File Storage Medium
Remote Compute Platform
Installed and brought into working state in response to a drag & drop operation

Figure 13

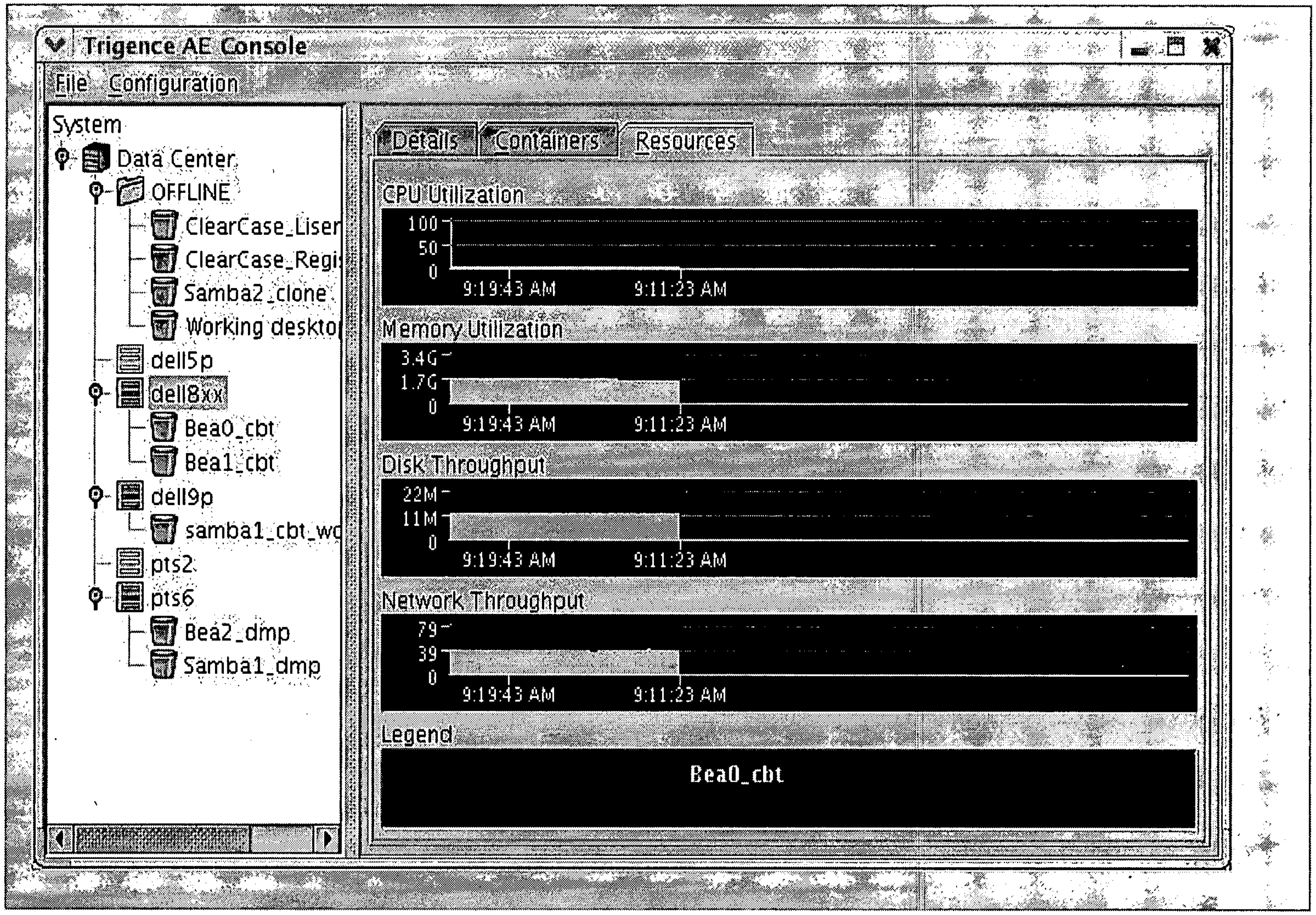

Trigence AE Console
File Configuration
System
Data Center
OFFLINE
ClearCase_Liser
ClearCase_Regi
Samba2_clone
Working deskto
dell5p
dell8xx
Bea0_cbt
Bea1_cbt
dell9p
samba1_cbt_wo
pts2
pts6
Bea2_dmp
Samba1_dmp
Details
Containers
Resources
CPU Utilization
100
50
0
9:19:43 AM
9:11:23 AM
Memory Utilization
3.4G
1.7G
0
9:19:43 AM
9:11:23 AM
Disk Throughput
22M
11M
0
9:19:43 AM
9:11:23 AM
Network Throughput
79
39
0
9:19:43 AM
9:11:23 AM
Legend
Bea0_cbt

Figure 14

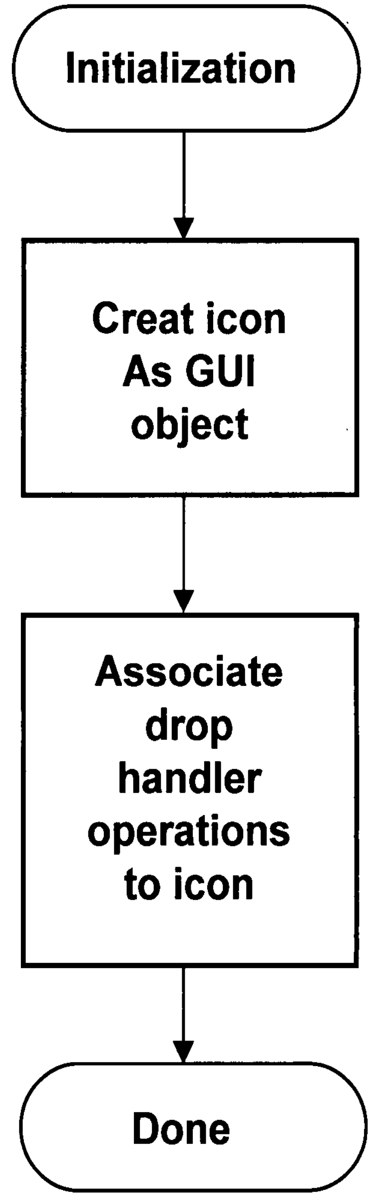

Initialization
Creat icon
As GUI
object
Associate
drop
handler
operations
to icon
Done

Figure 15

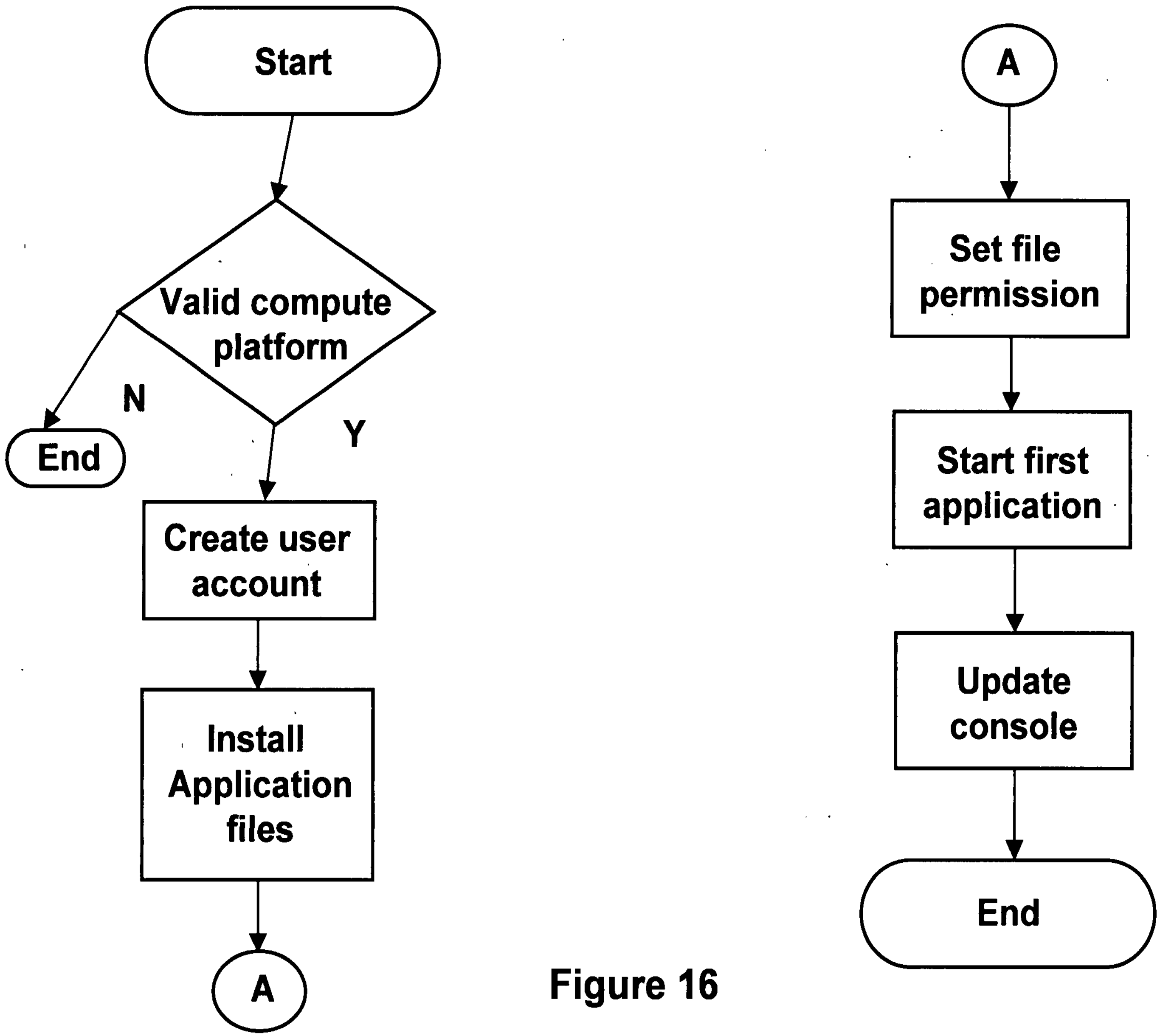

Start
Valid compute platform
N
End
Y
Create user account
Install Application files
A
A
Set file permission
Start first application
Update console
End

Figure 16

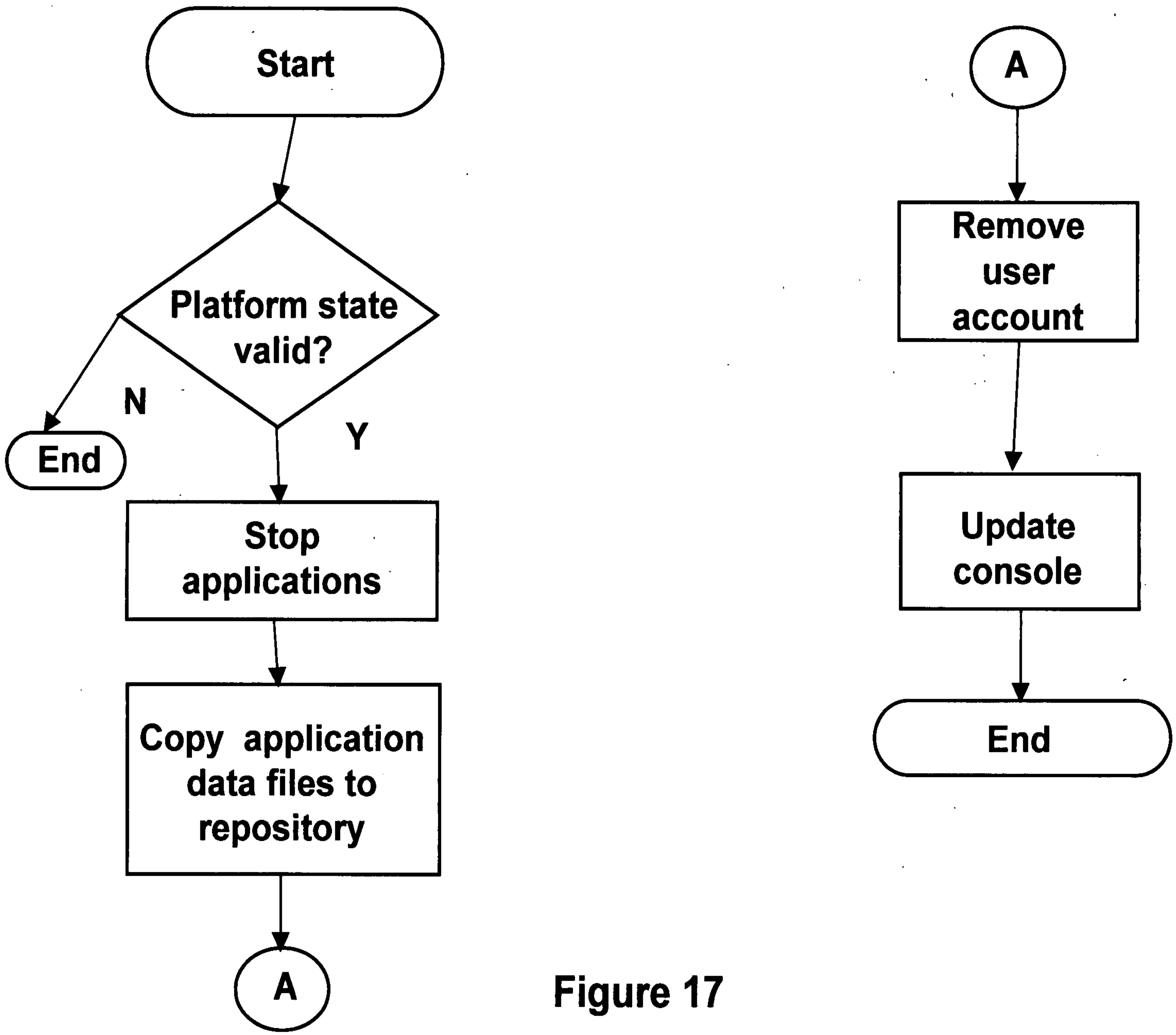

Start
Platform state valid?
N
End
Y
Stop applications
Copy application data files to repository
A
A
Remove user account
Update console
End

Figure 17

DECLARATION FOR PATENT APPLICATION

As below named inventors, we hereby declare that:

Our residence, post office addresses and citizenship are as stated below next to our names, **Donn ROCHETTE; Paul O'LEARY, and Dean HUFFMAN.**

We believe we are the original, first and joint inventors of the subject matter which is claimed and for which a patent is sought on the invention entitled

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

the specification of which is attached hereto.

I/we hereby authorize my/our attorney to insert here in parentheses (Application Number ____________________ , filed ____________________), the filing date and application number of said application, when known.

We hereby state that we have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

We acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of the Federal Regulations, S1.56(a).

We hereby claim the benefit of Title 35, U.S.C. 119(e), of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is now disclosed in the prior United States **provisional** patent applications

(Number)	(Filing Date)	(Status) patented/pending/abandoned
60/502,619	**September 15, 2003**	**Pending**
60/512,103	**October 20, 2003**	**Pending**

We hereby appoint the following agents to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith:

Address all telephone calls to: at telephone number:	**Charles E. Wands, Regn. No: 25,649** **(321) 725-4760**
Address all correspondence to:	**Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.** **1401 Citrus Center** **255 South Orange Avenue** **Box 3791** **Orlando, Florida** **USA 32802-3791**

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the U.S.C. and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of first inventor:	**Donn ROCHETTE**
Inventor's signature:	Donn Rochette Date: 9-1-99
Residence: Post office address:	**3408 30th Avenue, Fenton, Iowa 50539, U.S.A.** **As above**
Citizenship:	**U.S.**

Full name of second inventor:	**Paul O'LEARY**
Inventor's signature:	Date:
Residence: Post office address:	**22 Pentland Crescent, Kanata, Ontario, Canada K2K 1V5** **As above**
Citizenship:	**Canadian**

Full name of third inventor:	**Dean HUFFMAN**
Inventor's signature:	Date:
Residence: Post office address:	**13 Pelee Street, Kanata, Ontario, Canada K2M 2R4** **As above**
Citizenship:	**Canadian**

Address all telephone calls to: at telephone number:	**Charles E. Wands, Regn. No: 25,649 (321) 725-4760**
Address all correspondence to:	**Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.** **1401 Citrus Center** **255 South Orange Avenue** **Box 3791** **Orlando, Florida** **USA 32802-3791**

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the U.S.C. and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of first inventor: **Donn ROCHETTE**

Inventor's signature: Date:

Residence: **3408 30th Avenue, Fenton, Indiana 50539, U.S.A.**
Post office address: **As above**

Citizenship: **U.S.**

Full name of second inventor: **Paul O'LEARY**

Inventor's signature: Paul O'Leary Date: September 1, 2004

Residence: **22 Pentland Crescent, Kanata, Ontario, Canada K2K 1V5**
Post office address: **As above**

Citizenship: **Canadian**

Full name of third inventor: **Dean HUFFMAN**

Inventor's signature: [illegible] Date: September 1, 2004

Residence: **13 Pelée Street, Kanata, Ontario, Canada K2M 2R4**
Post office address: **As above**

Citizenship: **Canadian**

PATENT APPLICATION FEE DETERMINATION RECORD

Effective October 1, 2003

Application or Docket Number: 10939903

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	34	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	34 minus 20=	* 14
INDEPENDENT CLAIMS	2 minus 3 =	* 0
MULTIPLE DEPENDENT CLAIM PRESENT		☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐ RATE	FEE	OR	OTHER THAN SMALL ENTITY RATE	FEE
BASIC FEE	385.00	OR	BASIC FEE	770.00
XS 9=	126	OR	XS18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL	511	OR	TOTAL	

CLAIMS AS AMENDED - PART II

AMENDMENT A

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

SMALL ENTITY RATE	ADDITIONAL FEE	OR	OTHER THAN SMALL ENTITY RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE	OR	RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

RATE	ADDITIONAL FEE	OR	RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X43=		OR	X86=	
+145=		OR	+290=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

RECEIVED
CENTRAL FAX CENTER
OCT 17 2005

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE PATENT APPLICATION OF: **ROCHETTE ET AL.** OUR FILE NO: **78802 120-1 US**

SERIAL NUMBER: **10/939,903** GROUP: **2183**

FILED: **SEPTEMBER 13, 2004** CONFIRMATION NO: **5216**

TITLE: **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

Information Disclosure Statement

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Transmitted herewith is an Information Disclosure Statement (Form PTO-1449A) in the above-captioned application with references. In accordance with current USPTO procedures published 05 AUG 2003, in 1276 OG 55, copies of the U.S. patent documents cited in the form 1449A are not attached.

Certification

________ This Information Disclosure Statement is submitted within three months of:

(i) the filing date of the above-identified U.S .National Patent application, or
(ii) the date of entry into the U.S. National Stage of the above-identified International Application, or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies.

__X____ This Information Disclosure Statement is submitted prior to the mailing date of the first Office Action on the merits received by Applicant in the above-identified application.

________ This Information Disclosure Statement is submitted after three months from

(i) the filing date of the above-identified U.S. National Patent application, or
(ii) after three months from entry into the U.S. National Stage of the above-identified International Application; or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies; and after the mailing date of the first Office Action on the merits of the above-identified application, but prior to issuance of the earlier of any Final Action or Notice of Allowance sent in such application. The certification under 37 C.F.R. §1.97(e) is submitted separately or below, or the fee required under 37 C.F.R. §1.97(c) and § 1.17(p) is submitted herewith.

________ This Information Disclosure Statement is submitted after the earlier of the mailing date of a final rejection or Notice of Allowance sent in this application but before payment of the Issue Fee. The certification required under 37 C.F.R. § 1.97(e) is submitted separately or below. A petition to the Commissioner and the appropriate fee pursuant to §1.17(i) (1) are submitted herewith.

_______ A certification under 37 C.F.R §1.97 is submitted herewith separately from this paper.

_X_____ It is hereby certified that each item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this statement.

_______ No item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application or, to the knowledge of the undersigned, after making reasonable inquiry, was known to any individual designated in 37 C.F.R. §1.56(c) more than three months prior to the filing of this statement.

Respectfully submitted,

CHARLES E. WANDS
Reg. No. 25,649

Telephone: (321) 725-4760

Customer No.: 27975

CERTIFICATE OF FACSIMILE TRANSMISSION

I HEREBY CERTIFY that the foregoing correspondence has been forwarded via facsimile number 571-273-8300 to MAIL STOP AMENDMENT, COMMISSIONER FOR PATENTS, this 17 day of October 2005.

J. Kallemeres

Form PTO 1449A U.S. Department of Commerce Patent and Trademark Office Information Disclosure Statement by Applicant	ATTY. DOCKET NUMBER: 78802 120-1 US	SERIAL NUMBER: 10/939,903
	APPLICANT: ROCHETTE ET AL.	
	FILING DATE: SEPTEMBER 13, 2004	GROUP: 2183

U.S. Patent Documents

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILING APPL
		2002/0174215 A1	Nov. 21, 2002	Schaefer	709	224	

Foreign Patent Documents

		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION YES	TRANSLATION NO

Other Documents (Including Author, Title, Date Pertinent Pages, Etc.)

EXAMINER	DATE CONSIDERED

EXAMINER: Initial if citation is considered, whether or not citation is in conformance with MPEP 609; draw a line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Patent Application of: Rochette et al.		**Our File:**	**78802 (120-1 US)**
Serial No:	**10/939,903**	**Group:**	**2183**
Filed:	**September 13, 2004**	**Confirmation No.**	**5216**
Title:	**SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**		

Information Disclosure Statement

EFILED
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Transmitted herewith is an Information Disclosure Statement (Form PTO-1449A) in the above-captioned application with references. In accordance with current USPTO procedures published 05 AUG 2003, in 1276 OG 55, copies of the U.S. patent documents cited in the form 1449A are not attached.

Certification

________ This Information Disclosure Statement is submitted within three months of:

(i) the filing date of the above-identified U.S .National Patent application, or
(ii) the date of entry into the U.S. National Stage of the above-identified International Application, or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies.

__X__ This Information Disclosure Statement is submitted prior to the mailing date of the first Office Action on the merits received by Applicant in the above-identified application.

________ This Information Disclosure Statement is submitted after three months from

(i) the filing date of the above-identified U.S. National Patent application, or
(ii) after three months from entry into the U.S. National Stage of the above-identified International Application; or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies; and after the mailing date of the first Office Action on the merits of the above-identified application, but prior to issuance of the earlier of any Final Action or Notice of Allowance sent in such application. The certification under 37 C.F.R. §1.97(e) is submitted separately or below, or the fee required under 37 C.F.R. §1.97(c) and § 1.17(p) is submitted herewith.

________ This Information Disclosure Statement is submitted after the earlier of the mailing date of a final rejection or Notice of Allowance sent in this application but before payment of the Issue Fee. The certification required under 37 C.F.R. § 1.97(e) is submitted separately or below. A petition to the Commissioner and the appropriate fee pursuant to §1.17(i) (1) are submitted herewith.

_______ A certification under 37 C.F.R §1.97 is submitted herewith separately from this paper.

_______ It is hereby certified that each item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this statement.

_______ No item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application or, to the knowledge of the undersigned, after making reasonable inquiry, was known to any individual designated in 37 C.F.R. §1.56(c) more than three months prior to the filing of this statement.

Respectfully submitted,

NOV 28 2006

/charles edmund wands/
Reg. No: 25,649

Date:

CUSTOMER NO. 27975

Form PTO 1449A U.S. Department of Commerce Patent and Trademark Office Information Disclosure Statement by Applicant	ATTY. DOCKET NUMBER: 78802 (120-1 US)	SERIAL NUMBER: 10/939,903
	APPLICANT: ROCHETTE ET AL.	
	FILING DATE: September 13, 2004	GROUP: 2183

U.S. Patent Documents

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILIN APPR
		2002/0004854 A1	Jan 10, 2002	Hartley			
		2003/0101292 A1	May 20, 2003	Fisher			

Foreign Patent Documents

		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION	
							YES	NO
		WO 02/06941 A	Jan 24, 2002	Connectix Corporation				
		WO 06/039181 A	Apr 13, 2006	Citrix Systems, Inc.				

Other Documents (Including Author, Title, Date Pertinent Pages, Etc.)

EXAMINER	DATE CONSIDERED

EXAMINER: Initial if citation is considered, whether or not citation is in conformance with MPEP 609; draw a line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 April 2006 (13.04.2006)

PCT

(10) International Publication Number
WO 2006/039181 A1

(51) International Patent Classification[7]: G06F 9/46

(21) International Application Number: PCT/US2005/033994

(22) International Filing Date: 23 September 2005 (23.09.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

10/711,737	30 September 2004 (30.09.2004)	US
10/711,736	30 September 2004 (30.09.2004)	US
10/711,735	30 September 2004 (30.09.2004)	US
10/711,734	30 September 2004 (30.09.2004)	US
10/711,733	30 September 2004 (30.09.2004)	US
10/711,732	30 September 2004 (30.09.2004)	US
10/956,723	1 October 2004 (01.10.2004)	US
11/231,284	19 September 2005 (19.09.2005)	US
11/231,316	19 September 2005 (19.09.2005)	US
11/231,317	19 September 2005 (19.09.2005)	US
11/231,315	19 September 2005 (19.09.2005)	US
11/231,370	19 September 2005 (19.09.2005)	US

(71) **Applicant** *(for all designated States except US)*: **CITRIX SYSTEMS, INC.** [US/US]; 851 West Cypress Creek Road, Fort Lauderdale, FL 33309 (US).

(72) **Inventors; and**

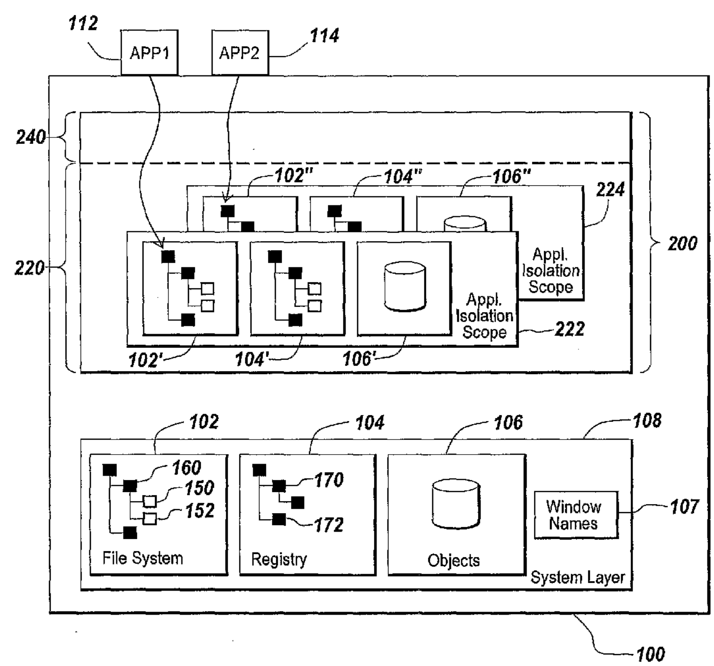
(75) **Inventors/Applicants** *(for US only)*: **LABORCZFALVI, Lee, George** [AU/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, Nort Ryde NSW 2113 (AU). **ROYCHOUDHRY, Anil** [AU/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde New South Wales 2113 (AU). **BORZYCKI, Andrew, Gerard** [AU/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde New South Wales 2113 (AU). **CHIN, Huai, Chiun** [MY/AU]; c/o Citrix systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde NSW 2113 (AU). **MAZZAFERRI, Richard, James** [AU/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde New South Wales 2113 (AU). **BISSETT, Nicholas, Alexander** [GB/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde New South Wales 2113 (AU). **MUIR, Jeffrey, Dale** [AU/AU]; c/o Citrix Systems Australasia R & D Pty Ltd., Level 3, 1 Julius Avenue, North Ryde New South Wales 2113 (AU).

(74) **Agent: LANZA, John, D.**; Choate, Hall & Stewart, Two International Place, Boston, MA 02110 (US).

(81) **Designated States** *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN,

[Continued on next page]

(54) **Title:** METHOD AND APPARATUS FOR MOVING PROCESSES BETWEEN ISOLATION ENVIRONMENTS



(57) **Abstract:** A method for moving an executing process from a source isolation scope to a target isolation scope includes the step of determining that the process is in a state suitable for moving. The association of the process changes from a source isolation scope to a target isolation scope. A rule loads in association with the target isolation scope.

WO 2006/039181 A1

WO 2006/039181 A1

CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR MOVING PROCESSES BETWEEN ISOLATION ENVIRONMENTS

Field of the Invention

The invention relates to managing execution of software applications by computers and, in particular, to methods and apparatus for moving processes between isolation enviroments.

Background of the Invention

Computer software application programs, during execution and installation, make use of a variety of native resources provided by the operating system of a computer. A traditional, single-user computer is depicted in FIG. 1A. As shown in FIG. 1A, native resources provided by the operating system 100 may include a file system 102, a registry database 104, and objects 106. The file system 102 provides a mechanism for an application program to open, create, read, copy, modify, and delete data files 150, 152. The data files 150, 152 may be grouped together in a logical hierarchy of directories 160, 162. The registry database 104 stores information regarding hardware physically attached to the computer, which system options have been selected, how computer memory is set up, various items of application-specific data, and what application programs should be present when the operating system 100 is started. As shown in FIG. 1A, the registry database 104 is commonly organized in a logical hierarchy of "keys" 170, 172 which are containers for registry values. The operating system 100 may also provide a number of communication and synchronization objects 106, including semaphores, sections, mutexes, timers, mutants, and pipes. Together, the file system 102, registry database 104, objects 106, and any other native resources made available by the operating system 100 will be referred to

throughout this document as the "system layer" 108. The resources provided by the system layer 108 are available to any application or system program 112, 114.

A problem arises, however, when execution or installation of two incompatible application programs 112, 114 is attempted. As shown in FIG. 1A, two application programs, APP1 112 and APP2 114, execute "on top of" the operating system 100, that is, the application programs make use of functions provided by the operating system to access native resources. The application programs are said to be incompatible with one another when they make use of native resources 102, 104, 106 in a mutually exclusive manner either during execution or during the installation process. APP1 112 may require, or attempt to install, a file located by the pathname c:\windows\system32\msvcrt.dll and APP2 114 may require, or attempt to install, a second, different file that is located by the same pathname. In this case, APP1 112 and APP2 114 cannot be executed on the same computer and are said to be incompatible with one another. A similar problem may be encountered for other native resources. This is, at best, inconvenient for a user of the computer who requires installation or execution of both APP1 112 and APP2 114 together in the same operating system 100 environment.

FIG. 1B depicts a multi-user computer system supporting concurrent execution of application programs 112, 114, 112', 114' on behalf of several users. As shown in FIG. 1B, a first instance of APP1 112 and a first instance of APP2 114 are executed in the context of a first user session 110, a second instance of APP1 112' is executed in the context of a second user session 120, and a second instance of APP2 114' is executed in the context of a third user session 130. In this environment, a problem arises if both instances of APP1 112, 112' and both instances of APP2 114, 114' make use of native resources 102, 104, 106 as if only a single user executes the application. For example, the APP1 112 may store application specific data in a registry key 170. When the

first instance of APP1 112 executing in the first user context 110 and the second instance of APP1 112' executing in the second user context 120 both attempt to store configuration data to the same registry key 170, incorrect configuration data will be stored for one of the users. A similar problem can occur for other native resources.

The present invention addresses these application program compatibility and sociability problems.

Summary of the Invention

The present invention allows installation and execution of application programs that are incompatible with each other, and incompatible versions of the same application program, on a single computer. In addition, it allows the installation and execution on a multi-user computer of programs that were created for a single-user computer or were created without consideration for issues that arise when executing on a multi-user computer. The methods and apparatus described are applicable to single-user computing environments, which includes environments in which multiple users may use a single computer one after the other, as well as multi-user computing environments, in which multiple users concurrently use a single computer. The present invention virtualizes user and application access to native resources, such as the file system, the registry database, system objects, window classes and window names, without modification of the application programs or the underlying operating system. In addition, virtualized native resources may be stored in native format (that is, virtualized files are stored in the file system, virtualized registry entries are stored in the registry database, etc.) allowing viewing and management of virtualized resources to be accomplished using standard tools and techniques. The provided isolation scopes may be taken advantage of by moving processes between isolation scopes, into isolation scopes, or out of isolation scopes while those processes are executing, allowing the view of native resources provided to those application to change.

In one aspect, the invention relates to a method for moving an executing process from a first isolation scope to a second isolation scope. A determination is made that the process is in a state suitable for moving. The association of the process changes from a first isolation scope to a second isolation scope. A rule loads in association with the second isolation scope.

In one embodiment, determining whether the process is in a state suitable for moving is accomplished by monitoring whether the process is processing a request. In some embodiments, the process is put in a state suitable for moving. In another embodiment, the association of the process from the first isolation scope to the second isolation scope changes in a file system filter driver. In yet another embodiment, the association of the process from the first isolation scope to the second isolation scope changes in one of a kernel hooking function and a user mode hooking function.

In another aspect, a method relates to moving an executing process into an isolation scope. A determination is made that the process is in a state suitable for moving. There is an association of the process with an isolation scope. A rule loads in association with the isolation scope.

In one embodiment, determining whether the process is in a state suitable for moving is accomplished by monitoring whether the process is processing a request. In some embodiments, the process is put in a state suitable for moving. In other embodiments, information associating the process with the isolation scope is written to a rules engine.

<u>Brief Description of the Drawings</u>

The invention is pointed out with particularity in the appended claims. The advantages of the invention described above, as well as further advantages of the invention, may be better understood by reference to the following description taken in conjunction with the accompanying drawings, in which:

FIG. 1A is a block diagram of a prior art operating system environment supporting execution of two application programs on behalf of a user;

FIG. 1B is a block diagram of a prior art operating system environment supporting concurrent execution of multiple applications on behalf of several users;

FIG. 2A is a block diagram of an embodiment of a computer system having reduced application program compatibility and sociability problems;

FIG. 2B is a diagram of an embodiment of a computer system having reduced application program compatibility and sociability problems;

FIG. 2C is a flowchart showing one embodiment of steps taken to associate a process with an isolation scope;

FIG 3A is a flowchart showing one embodiment of steps taken to virtualize access to native resources in a computer system;

FIG. 3B is a flowchart showing an embodiment of steps taken to identify a replacement instance in execute mode;

FIG. 3C is a flowchart depicting one embodiment of steps taken in install mode to identify a literal resource when a request to open a native resource is received that indicates the resource is being opened with the intention of modifying it;

FIG. 3D is a flowchart depicting one embodiment of steps taken in install mode to identify the literal resource when a request to create a virtual resource is received;

FIG. 4 is a flowchart depicting one embodiment of the steps taken to open an entry in a file system in the described virtualized environment;

FIG. 5 is a flowchart depicting one embodiment of the steps taken to delete an entry from a file system in the described virtualized environment;

FIG. 6 is a flowchart depicting one embodiment of the steps taken to enumerate entries in a file system in the described virtualized environment;

FIG. 7 is a flowchart depicting one embodiment of the steps taken to create an entry in a file system in the described virtualized environment;

FIG. 8 is a flowchart depicting one embodiment of the steps taken to open a registry key in the described virtualized environment;

FIG. 9 is a flowchart depicting one embodiment of the steps taken to delete a registry key in the described virtualized environment;

FIG. 10 is a flowchart depicting one embodiment of the steps taken to enumerate subkeys of a key in a registry database in the described virtualized environment;

FIG. 11 is a flowchart depicting one embodiment of the steps taken to create a registry key in the described virtualized environment;

FIG. 12 is a flowchart depicting one embodiment of the steps taken to virtualize access to named objects;

FIG. 13 is a flowchart depicting one embodiment of the steps taken to virtualize window names and window classes in the described environment;

FIGs. 13A is a flowchart depicting one embodiment of the steps taken to determine literal window names and window class names;

FIG. 14 is a flowchart depicting one embodiment of the steps taken to invoke an out-of-process COM server in the described virtualized environment;

FIG. 15 is a flowchart depicting one embodiment of the steps taken to virtualize application invocation using file-type association; and

FIG. 16 is a flowchart depicting one embodiment of the steps taken to move a process from a source isolation scope to a target isolation scope.

Index

The index is intended to help the reader follow the discussion of the invention:

Detailed Description of the Invention

1.0 Isolation Environment Conceptual Overview

1.1 Application Isolation

Referring now to FIG. 2A, one embodiment of a computer running under control of an operating system 100 that has reduced application compatibility and application sociability problems is shown. The operating system 100 makes available various native resources to application programs 112, 114 via its system layer 108. The view of resources embodied by the system layer 108 will be termed the "system scope". In order to avoid conflicting access to native resources 102, 104, 106, 107 by the application programs 112, 114, an isolation environment 200 is provided. As shown in FIG. 2A, the isolation environment 200 includes an application isolation layer 220 and a user isolation layer 240. Conceptually, the isolation environment 200 provides, via the application isolation layer 220, an application program 112, 114, with a unique view of native resources, such as the file system 102, the registry 104, objects 106, and window names 107. Each isolation layer modifies the view of native resources provided to an application. The modified view of native resources provided by a layer will be referred to as that layer's "isolation scope". As shown in FIG. 2A, the application isolation layer includes two application isolation scopes 222, 224. Scope 222 represents the view of native resources provided to application 112 and scope 224 represents the view of native resources provided to application 114. Thus, in the embodiment shown in FIG. 2A, APP1 112 is provided with a specific view of the file system 102', while APP2 114 is provided with another view of the file system 102" which is specific to it. In some embodiments, the application isolation layer 220 provides a specific view of native resources 102, 104, 106, 107 to each individual application program executing on top of the operating system 100. In other embodiments, application programs 112, 114 may be grouped into sets and, in these embodiments, the application isolation layer 220 provides a specific view of native resources for each set of application

programs. Conflicting application programs may be put into separate groups to enhance the compatibility and sociability of applications. In still further embodiments, the applications belonging to a set may be configured by an administrator. In some embodiments, a "passthrough" isolation scope can be defined which corresponds exactly to the system scope. In other words, applications executing within a passthrough isolation scope operate directly on the system scope.

In some embodiments, the application isolation scope is further divided into layered sub-scopes. The main sub-scope contains the base application isolation scope, and additional sub-scopes contain various modifications to this scope that may be visible to multiple executing instances of the application. For example, a sub-scope may contain modifications to the scope that embody a change in the patch level of the application or the installation or removal of additional features. In some embodiments, the set of additional sub-scopes that are made visible to an instance of the executing application is configurable. In some embodiments, that set of visible sub-scopes is the same for all instances of the executing application, regardless of the user on behalf of which the application is executing. In others, the set of visible sub-scopes may vary for different users executing the application. In still other embodiments, various sets of sub-scopes may be defined and the user may have a choice as to which set to use. In some embodiments, sub-scopes may be discarded when no longer needed. In some embodiments, the modifications contained in a set of sub-scopes may be merged together to form a single sub-scope.

1.2 User Isolation

Referring now to FIG. 2B, a multi-user computer having reduced application compatibility and application sociability problems is depicted. The multi-user computer includes native resources 102, 104, 106, 107 in the system layer 108, as well as the isolation environment 200 discussed immediately above. The application isolation layer 220 functions as discussed above,

providing an application or group of applications with a modified view of native resources. The user isolation layer 240, conceptually, provides an application program 112, 114, with a view of native resources that is further altered based on user identity of the user on whose behalf the application is executed. As shown in FIG. 2B, the user isolation layer 240 may be considered to comprise a number of user isolation scopes 242', 242", 242''', 242'''', 242''''', 242'''''' (generally 242). A user isolation scope 242 provides a user-specific view of application-specific views of native resources. For example, APP1 112 executing in user session 110 on behalf of user "a" is provided with a file system view 102'(a) that is altered or modified by both the user isolation scope 242' and the application isolation scope 222.

Put another way, the user isolation layer 240 alters the view of native resources for each individual user by "layering" a user-specific view modification provided by a user isolation scope 242' "on top of" an application-specific view modification provided by an application isolation scope 222, which is in turn "layered on top of" the system-wide view of native resources provided by the system layer. For example, when the first instance of APP1 112 accesses an entry in the registry database 104, the view of the registry database specific to the first user session and the application 104'(a) is consulted. If the requested registry key is found in the user-specific view of the registry 104'(a), that registry key is returned to APP1 112. If not, the view of the registry database specific to the application 104' is consulted. If the requested registry key is found in the application-specific view of the registry 104', that registry key is returned to APP1 112. If not, then the registry key stored in the registry database 104 in the system layer 108 (i.e. the native registry key) is returned to APP1 112.

In some embodiments, the user isolation layer 240 provides an isolation scope for each individual user. In other embodiments, the user isolation layer 240 provides an isolation scope for a group of users, which may be defined by roles within the organization or may be predetermined by an administrator. In

still other embodiments, no user isolation layer 240 is provided. In these embodiments, the view of native resources seen by an application program is that provided by the application isolation layer 220. The isolation environment 200, although described in relation to multi-user computers supporting concurrent execution of application programs by various users, may also be used on single-user computers to address application compatibility and sociability problems resulting from sequential execution of application programs on the same computer system by different users, and those problems resulting from installation and execution of incompatible programs by the same user.

In some embodiments, the user isolation scope is further divided into sub-scopes. The modifications by the user isolation scope to the view presented to an application executing in that scope is the aggregate of the modifications contained within each sub-scope in the scope. Sub-scopes are layered on top of each other, and in the aggregate view modifications to a resource in a higher sub-scope override modifications to the same resource in lower layers.

In some of these embodiments, one or more of these sub-scopes may contain modifications to the view that are specific to the user. In some of these embodiments, one or more sub-scopes may contain modifications to the view that are specific to sets of users, which may be defined by the system administrators or defined as a group of users in the operating system. In some of these embodiments, one of these sub-scopes may contain modifications to the view that are specific to the particular login session, and hence that are discarded when the session ends. In some of these embodiments, changes to native resources by application instances associated with the user isolation scope always affects one of these sub-scopes, and in other embodiments those changes may affect different sub-scopes depending on the particular resource changed.

1.3 Aggregate Views of Native Resources

The conceptual architecture described above allows an application executing on behalf of a user to be presented with an aggregate, or unified, virtualized view of native resources, specific to that combination of application and user. This aggregated view may be referred to as the "virtual scope". The application instance executing on behalf of a user is presented with a single view of native resources reflecting all operative virtualized instances of the native resources. Conceptually this aggregated view consists firstly of the set of native resources provided by the operating system in the system scope, overlaid with the modifications embodied in the application isolation scope applicable to the executing application, further overlaid with the modifications embodied in the user isolation scope applicable to the application executing on behalf of the user. The native resources in the system scope are characterized by being common to all users and applications on the system, except where operating system permissions deny access to specific users or applications. The modifications to the resource view embodied in an application isolation scope are characterized as being common to all instances of applications associated with that application isolation scope. The modifications to the resource view embodied in the user isolation scope are characterized as being common to all applications associated with the applicable application isolation scope that are executing on behalf of the user associated with the user isolation scope.

This concept can be extended to sub-scopes; the modifications to the resource view embodied in a user sub-scope are common to all applications associated with the applicable isolation sub-scope executing on behalf of a user, or group of users, associated with a user isolation sub-scope. Throughout this description it should be understood that whenever general reference is made to "scope," it is intended to also refer to sub-scopes, where those exist.

When an application requests enumeration of a native resource, such as a portion of the file system or registry database, a virtualized enumeration is constructed by first enumerating the "system-scoped" instance of the native

resource, that is, the instance found in the system layer, if any. Next, the "application-scoped" instance of the requested resource, that is the instance found in the appropriate application isolation scope, if any, is enumerated. Any enumerated resources encountered in the application isolation scope are added to the view. If the enumerated resource already exists in the view (because it was present in the system scope, as well), it is replaced with the instance of the resource encountered in the application isolation scope. Similarly, the "user-scoped" instance of the requested resource, that is the instance found in the appropriate user isolation scope, if any, is enumerated. Again, any enumerated resources encountered in the user isolation scope are added to the view. If the native resource already exists in the view (because it was present in the system scope or in the appropriate application isolation scope), it is replaced with the instance of the resource encountered in the user isolation scope. In this manner, any enumeration of native resources will properly reflect virtualization of the enumerated native resources. Conceptually the same approach applies to enumerating an isolation scope that comprises multiple sub-scopes. The individual sub-scopes are enumerated, with resources from higher sub-scopes replacing matching instances from lower sub-scopes in the aggregate view.

In other embodiments, enumeration may be performed from the user isolation scope layer down to the system layer, rather than the reverse. In these embodiments, the user isolation scope is enumerated. Then the application isolation scope is enumerated and any resource instances appearing in the application isolation scope that were not enumerated in the user isolation scope are added to the aggregate view that is under construction. A similar process can be repeated for resources appearing only in the system scope.

In still other embodiments, all isolation scopes may be simultaneously enumerated and the respective enumerations combined.

If an application attempts to open an existing instance of a native resource with no intent to modify that resource, the specific instance that is returned to the

application is the one that is found in the virtual scope, or equivalently the instance that would appear in the virtualized enumeration of the parent of the requested resource. From the point of view of the isolation environment, the application is said to be requesting to open a "virtual resource", and the particular instance of native resource used to satisfy that request is said to be the "literal resource" corresponding to the requested resource.

If an application executing on behalf of a user attempts to open a resource and indicates that it is doing so with the intent to modify that resource, that application instance is normally given a private copy of that resource to modify, as resources in the application isolation scope and system scope are common to applications executing on behalf of other users. Typically a user-scoped copy of the resource is made, unless the user-scoped instance already exists. The definition of the aggregate view provided by a virtual scope means that the act of copying an application-scoped or system-scoped resource to a user isolation scope does not change the aggregate view provided by the virtual scope for the user and application in question, nor for any other user, nor for any other application instance. Subsequent modifications to the copied resource by the application instance executing on behalf of the user do not affect the aggregate view of any other application instance that does not share the same user isolation scope. In other words, those modifications do not change the aggregate view of native resources for other users, or for application instances not associated with the same application isolation scope.

1.4 Association of processes with isolation scopes

Applications may be installed into a particular isolation scope (described below in more detail). Applications that are installed into an isolation scope are always associated with that scope. Alternatively, applications may be launched into a particular isolation scope, or into a number of isolation scopes. In effect, an application is launched and associated with one or more isolation scopes. The associated isolation scope, or scopes, provide the process with a particular

view of native resources. Applications may also be launched into the system scope, that is, they may be associated with no isolation scope. This allows for the selective execution of operating system applications such as Internet Explorer, as well as third party applications, within an isolation environment.

This ability to launch applications within an isolation scope regardless of where the application is installed mitigates application compatibility and sociability issues without requiring a separate installation of the application within the isolation scope. The ability to selectively launch installed applications in different isolation scopes provides the ability to have applications which need helper applications (such as Word, Notepad, etc.) to have those helper applications launched with the same rule sets.

Further, the ability to launch an application within multiple isolated environments allows for better integration between isolated applications and common applications.

Referring now to FIG. 2C, and in brief overview, a method for associating a process with an isolation scope includes the steps of launching the process in a suspended state (step 282). The rules associated with the desired isolation scope are retrieved (step 284) and an identifier for the process and the retrieved rules are stored in a memory element (step 286) and the suspended process is resumed (step 288). Subsequent calls to access native resources made by the process are intercepted or hooked (step 290) and the rules associated with the process identifier, if any, are used to virtualize access to the requested resource (step 292).

Still referring to FIG. 2C, and in more detail, a process is launched in a suspended state (step 282). In some embodiments, a custom launcher program is used to accomplish this task. In some of these embodiments, the launcher is specifically designed to launch a process into a selected isolation scope. In other embodiments, the launcher accepts as input a specification of the desired isolation scope, for example, by a command line option.

The rules associated with the desired isolation scope are retrieved (step 284). In some embodiments, the rules are retrieved from a persistent storage element, such as a hard disk drive or other solid state memory element. The rules may be stored as a relational database, flat file database, tree-structured database, binary tree structure, or other persistent data structure. In other embodiments, the rules may be stored in a data structure specifically configured to store them.

An identifier for the process, such as a process id (PID), and the retrieved rules are stored in a memory element (step 286). In some embodiments, a kernel mode driver is provided that receives operating system messages concerning new process creation. In these embodiments, the PID and the retrieved rules may be stored in the context of the driver. In other embodiments, a file system filter driver, or mini-filter, is provided that intercepts native resource requests. In these embodiments, the PID and the retrieved rules may be stored in the filter. In other embodiments still, all interception is performed by user-mode hooking and no PID is stored at all. The rules are loaded by the user-mode hooking apparatus during the process initialization, and no other component needs to know the rules that apply to the PID because rule association is performed entirely in-process.

The suspended process is resumed (step 288) and subsequent calls to access native resources made by the process are intercepted or hooked (step 290) and the rules associated with the process identifier, if any, are used to virtualize access to the requested resource (step 292). In some embodiments, a file system filter driver, or mini-filter, intercepts requests to access native resources and determines if the process identifier associated with the intercepted request has been associated with a set of rules. If so, the rules associated with the stored process identifier are used to virtualize the request to access native resources. If not, the request to access native resources is passed through unmodified. In other embodiments, a dynamically-linked library is loaded into the

newly-created process and the library loads the isolation rules. In still other embodiments, both kernel mode techniques (hooking, filter driver, mini-filter) and user-mode techniques are used to intercept calls to access native resources. For embodiments in which a file system filter driver stores the rules, the library may load the rules from the file system filter driver.

Processes that are "children" of processes associated with isolation scopes are associated with the isolation scopes of their "parent" process. In some embodiments, this is accomplished by a kernel mode driver notifying the file system filter driver when a child process is created. In these embodiments, the file system filter driver determines if the process identifier of the parent process is associated with an isolation scope. If so, file system filter driver stores an association between the process identifier for the newly-created child process and the isolation scope of the parent process. In other embodiments, the file system filter driver can be called directly from the system without use of a kernel mode driver. In other embodiments, in processes that are associated with isolation scopes, operating system functions that create new processes are hooked or intercepted. When request to create a new process are received from such a process, the association between the new child process and the isolation scope of the parent is stored.

In some embodiments, a scope or sub-scope may be associated with an individual thread instead of an entire process, allowing isolation to be performed on a per-thread basis. In some embodiments, per-thread isolation may be used for Services and COM+ servers.

1.4.1 Associating out-of-scope processes with isolation scopes

Another aspect of this invention is the ability to associate any application instance with any application isolation scope, regardless of whether the application was installed into that application isolation scope, another application isolation scope or no application isolation scope. Applications which were not installed into a particular application scope can nevertheless be executed on

behalf of a user in the context of an application isolation scope and the corresponding user isolation scope, because their native resources are available to them via the aggregated virtual scope formed by the user isolation scope, application isolation scope and system scope. Where it is desired to run an application in an isolation scope, this provides the ability for applications installed directly into the system scope to be run within the isolation scope without requiring a separate installation of the application within the isolation scope. This also provides the ability for applications installed directly into the system scope to be used as helper applications in the context of any isolation scope.

Each application instance, including all of the processes that make up the executing application, is associated with either zero or one application isolation scopes, and by extension exactly zero or one corresponding user isolation scopes. This association is used by the rules engine when determining which rule, if any, to apply to a resource request. The association does not have to be to the application isolation scope that the application was installed into, if any. Many applications that are installed into an isolation scope will not function correctly when running in a different isolation scope or no isolation scope because they cannot find necessary native resources. However, because an isolation scope is an aggregation of resource views including the system scope, an application installed in the system scope can generally function correctly inside any application isolation scope. This means that helper programs, as well as out-of-process COM servers, can be invoked and executed by an application executing on behalf of a user in a particular isolation scope.

In some embodiments, applications that are installed in the system scope are executed in an isolation scope for the purpose of identifying what changes are made to the computer's files and configuration settings as a result of this execution. As all affected files and configuration settings are isolated in the user isolation scope, these files and configuration settings are easily identifiable. In some of these embodiments, this is used in order to report on the changes made

to the files and configuration settings by the application. In some embodiments, the files and configuration settings are deleted at the end of the application execution, which effectively ensures that no changes to the computer's files and configuration setting are stored as a result of application execution. In still other embodiments, the files and configuration settings are selectively deleted or not deleted at the end of the application execution, which effectively ensures that only some changes to the computer's files and configuration setting are stored as a result of application execution.

2.0 Virtualization Mechanism Overview

Referring now to FIG. 3A, one embodiment of the steps to be taken to virtualize access to native resources in execute mode, which will be distinguished from install mode below, is shown. In brief overview, a request to access a native resource is intercepted or received (step 302). The request identifies the native resource to which access is sought. The applicable rule regarding how to treat the received access request is determined (step 304). If the rule indicates the request should be ignored, the access request is passed without modification to the system layer (step 306) and the result returned to the requestor (step 310). If the rule indicates that the access request should be either redirected or isolated, the literal instance of the resource that satisfies the request is identified (step 308), a modified or replacement request for the literal resource is passed to the system layer (step 306) and the result is returned to the requestor (step 310).

Still referring to FIG. 3, and in more detail, a request identifying a native resource is intercepted or received (step 302). In some embodiments, requests for native resources are intercepted by "hooking" functions provided by the operating system for applications to make native resource requests. In specific embodiments, this is implemented as a dynamically-linked library that is loaded into the address space of every new process created by the operating system, and which performs hooking during its initialization routine. Loading a DLL into every process may be achieved via a facility provided by the operating system, or

alternatively by modifying the executable image's list of DLLs to import, either in the disk file, or in memory as the executable image for the process is loaded from disk. In other embodiments, function hooking is performed by a service, driver or daemon. In other embodiments, executable images, including shared libraries and executable files, provided by the operating system may be modified or patched in order to provide function hooks or to directly embody the logic of this invention. For specific embodiments in which the operating system is a member of the Microsoft WINDOWS family of operating systems, interception may be performed by a kernel mode driver hooking the System Service Dispatch Table. In still other embodiments, the operating system may provide a facility allowing third parties to hook functions that request access to native resources. In some of these embodiments, the operating system may provide this facility via an application programming interface (API) or a debug facility.

In other embodiments, the native resource request is intercepted by a filter in the driver stack or handler stack associated with the native resource. For example, some members of the family of Microsoft WINDOWS operating systems provide the capability to plug a third-party filter driver or mini-filter into the file system driver stack and a file system filter driver or mini-filter may be used to provide the isolation functions described below. In still other embodiments the invention comprises a file system implementation that directly incorporates the logic of this invention. Alternatively, the operating system may be rewritten to directly provide the functions described below. In some embodiments, a combination of some or all of the methods listed above to intercept or receive requests for resources may be simultaneously employed.

In many embodiments, only requests to open an existing native resource or create a new native resource are hooked or intercepted. In these embodiments, the initial access to a native resource is the access that causes the resource to be virtualized. After the initial access, the requesting application program is able to communicate with the operating system concerning the

virtualized resource using a handle or pointer or other identifier provided by the operating system that directly identifies the literal resource. In other embodiments, other types of requests to operate on a virtualized native resource are also hooked or intercepted. In some of these embodiments, requests by the application to open or create virtual resources return virtual handles that do not directly identify the literal resource, and the isolation environment is responsible for translating subsequent requests against virtual handles to the corresponding literal resource. In some of those embodiments, additional virtualization operations can be deferred until proven necessary. For example, the operation of providing a private modifiable copy of a resource to an isolation scope can be deferred until a request to change the resource is made, rather than when the resource is opened in a mode that allows subsequent modification.

Once the native resource request is intercepted or received, the applicable rule determining how to treat the particular request is determined (step 304). The most applicable rule may be determined by reference to a rules engine, a database of rules, or a flat file containing rules organized using an appropriate data structure such as a list or a tree structure. In some embodiments, rules are accorded a priority that determines which rule will be regarded as most applicable when two or more rules apply. In some of these embodiments, rule priority is included in the rules themselves or, alternatively, rule priority may be embedded in the data structure used to store the rules, for example, rule priority may be indicated by a rule's position in a tree structure. The determined rule may include additional information regarding how to process the virtualized resource request such as, for example, to which literal resource to redirect the request. In a specific embodiment a rule is a triple comprising a filter field, an action field, and data field. In this embodiment, the filter field includes the filter used to match received native resource requests to determine if the rule is valid for the requested resource name. The action field can be "ignore," "redirect," or "isolate". The data field may be any additional information concerning the action

to be taken when the rule is valid, including the function to be used when the rule is valid.

A rule action of "ignore" means the request directly operates on the requested native resource in the system scope. That is, the request is passed unaltered to the system layer 108 (step 306) and the request is fulfilled as if no isolation environment 200 exists. In this case, the isolation environment is said to have a "hole", or the request may be referred to as a "passthrough" request.

If the rule action indicates that the native resource request should be redirected or isolated, then the literal resource that satisfies the request is identified (step 308).

A rule action of "redirect" means that the request directly operates on a system-scoped native resource, albeit a different resource than specified in the request. The literal resource is identified by applying a mapping function specified in, or implied by, the data field of the determined rule to the name of the requested native resource. In the most general case the literal native resource may be located anywhere in the system scope. As a simple example, the rule {prefix_match ("c:\temp\", resource name), REDIRECT, replace_prefix ("c:\temp\", "d:\wutemp\", resource name)} will redirect a requested access to the file c:\temp\examples\d1.txt to the literal file d:\wutemp\examples\d1.txt. The mapping function included in the data field of the rule and the matching function may be further generalized to support complex behaviors by, for example, using regular expressions. Some embodiments may provide the ability to specify mapping functions that locate the literal resource within the user isolation scope or a sub-scope applicable to the application executing on behalf of the user, or the application isolation scope or a sub-scope applicable to the application. Further embodiments may provide the ability to specify mapping functions that locate the literal resource within the application isolation scope applicable to a different application in order to provide a controlled form of interaction between isolated applications. In some particular embodiments, the "redirect" action can

be configured to provide behavior equivalent to the "ignore" rule action. In these embodiments, the literal resource is exactly the requested native resource. When this condition is configured, the isolation environment may be said to have a "hole," or the request may be referred to as a "passthrough" request.

A rule action of "isolate" means that the request operates on a literal resource that is identified using the appropriate user isolation scope and application isolation scope. That is, the identifier for the literal resource is determined by modifying the identifier for the requested native resource using the user isolation scope, the application isolation scope, both scopes, or neither scope. The particular literal resource identified depends on the type of access requested and whether instances of the requested native resource already exist in the applicable user isolation scope, the applicable application isolation scope and the system scope.

FIG. 3B depicts one embodiment of steps taken to identify the literal resource (step 306 in FIG. 3A) when a request to open a native resource is received that indicates the resource is being opened with the intention of modifying it. Briefly, a determination is made whether the user-scoped instance, that is, an instance that exists in an applicable user scope or user sub-scope, of the requested native resource exists (step 354). If so, the user-scoped instance is identified as the literal resource for the request (step 372), and that instance is opened and returned to the requestor. If the user-scoped instance does not exist, a determination whether the application-scoped instance of the requested native resource exists is made (step 356). If the application-scoped instance exists, it is identified as the "candidate" resource instance (step 359), and permission data associated with the candidate instance is checked to determine if modification of that instance is allowed (step 362). If no application-scoped instance exists, then a determination is made whether the system-scoped instance of the requested native resource exists (step 358). If it does not, an error condition is returned to the requestor indicating that the requested

virtualized resource does not exist in the virtual scope (step 360). However, if the system-scoped resource exists, it is identified as the candidate resource instance (step 361), and permission data associated with the candidate instance is checked to determine if modification of that instance is allowed (step 362). If not, an error condition is returned to the requestor (step 364) indicating that modification of the virtualized resource is not allowed. If the permission data indicates that the candidate resource may be modified, a user-scoped copy of the candidate instance of the native resource is made (step 370), the user-scoped instance is identified as the literal instance for the request (step 372), and is opened and returned to the requestor.

Still referring to FIG. 3B, and in more detail, a determination is made whether the user-scoped resource exists, or in other words, whether the requested resource exists in the applicable user scope or sub-scope (step 354). The applicable user scope or sub-scope is the scope associated with the user that is layered on the application isolation scope associated with the application making the request. The user isolation scope or a sub-scope, in the file system case, may be a directory under which all files that exist in the user isolation scope are stored. In some of these embodiments, the directory tree structure under the user isolation directory reflects the path of the requested resource. For example, if the requested file is c:\temp\test.txt and the user isolation scope directory is d:\user1\app1\, then the path to the user-scoped literal file may be d:\user1\app1\c\temp\test.txt. In other embodiments, the path to the user-scoped literal may be defined in a native naming convention. For example, the path to the user-scoped literal file may be d:\user1\app1\device\harddisk1\temp\test.txt. In still other embodiments, the user-scoped files may all be stored in a single directory with names chosen to be unique and a database may be used to store the mapping between the requested file name and the name of the corresponding literal file stored in the directory. In still other embodiments, the contents of the literal files may be

stored in a database. In still other embodiments, the native file system provides the facility for a single file to contain multiple independent named "streams", and the contents of the user-scoped files are stored as additional streams of the associated files in the system scope. Alternatively, the literal files may be stored in a custom file system that may be designed to optimize disk usage or other criteria of interest.

If the user-scoped resource instance does not exist, then a determination is made whether the application-scoped resource exists, or in other words whether the requested resource exists in the application isolation scope (step 356). The methods described above are used to make this determination. For example, if the requested file is c:\temp\test.txt and the application isolation scope directory is e:\app1\, then the path to the application-scoped file may be e:\app1\c\temp\test.txt. As above, the path to the application-scoped file may be stored in a native naming convention. The embodiments described above may also apply to the application isolation scope.

If the application-scoped resource does not exist, then a determination is made whether the system-scoped resource exists, or in other words, whether the requested resource exists in the system scope (step 358). For example, if the requested file is c:\temp\test.txt then the path to the system-scoped file is c:\temp\test.txt. If the requested resource does not exist in the system scope, an indication that the requested resource does not exist in the virtual scope is returned to the requestor (step 360).

Whether the candidate resource instance for the requested resource is located in the application isolation scope or in the system scope, a determination is made whether modification of the candidate resource instance is allowed (step 362). For example, the candidate native resource instance may have associated native permission data indicating that modification of the candidate instance is not allowed by that user. Further, the rules engine may include configuration settings instructing the isolation environment to obey or override the native

permission data for virtualized copies of resources. In some embodiments, the rules may specify for some virtual resources the scope in which modifications are to occur, for example the system scope or the application isolation scope or a sub-scope, or the user isolation scope or a sub-scope. In some embodiments, the rules engine may specify configuration settings that apply to subsets of the virtualized native resources based on hierarchy or on type of resource accessed. In some of these embodiments, the configuration settings may be specific to each atomic native resource. In another example, the rules engine may include configuration data that prohibits or allows modification of certain classes of files, such as executable code or MIME types or file types as defined by the operating system.

If the determination in step 362 is that modification of the candidate resource instance is not allowed, then an error condition is returned to the requestor indicating that write access to the virtual resource is not allowed (step 364). If the determination in step 362 is that modification of the candidate resource instance is allowed, then the candidate instance is copied to the appropriate user isolation scope or sub-scope (step 370). For embodiments in which the logical hierarchy structure of the requested native resource is maintained in the isolation scopes, copying the candidate instance of the resource to the user isolation scope may require the creation in the user isolation scope of hierarchy placeholders. A hierarchy placeholder is a node that is placed in the hierarchy to correctly locate the copied resource in the isolation scope. A hierarchy placeholder stores no data, is identified as a placeholder node, and “does not exist” in the sense that it cannot be the literal resource returned to a requestor. In some embodiments, the identification of a node as a placeholder node is made by recording the fact in metadata attached to the node, or to the parent of the node, or to some other related entity in the system layer. In other embodiments, a separate repository of placeholder node names is maintained.

In some embodiments, the rules may specify that modifications to particular resources may be made at a particular scope, such as the application isolation scope. In those cases the copy operation in step 370 is expanded to determine whether modification to the candidate resource instance is allowed at the scope or sub-scope in which it is found. If not, the candidate resource instance is copied to the scope or sub-scope in which modification is allowed, which may not always be the user isolation scope, and the new copy is identified as the literal resource instance (step 372). If so, the candidate resource instance is identified as the literal instance (step 372), and is opened and the result returned to the requestor (step 306).

Referring back to FIG. 3A, the literal resource instance, whether located in step 354 or created in step 370, is opened (step 306) and returned to the requestor (step 310). In some embodiments, this is accomplished by issuing an "open" command to the operating system and returning to the requestor the response from the operating system to the "open" command.

If an application executing on behalf of a user deletes a native resource, the aggregated view of native resources presented to that application as the virtual scope must reflect the deletion. A request to delete a resource is a request for a special type of modification, albeit one that modifies the resource by removing its existence entirely. Conceptually a request to delete a resource proceeds in a similar manner to that outlined in Figure 3A, including the determination of the literal resource as outlined in Figure 3B. However, step 306 operates differently for isolated resources and for redirected or ignored resources. For redirect and ignore, the literal resource is deleted from the system scope. For isolate, the literal resource is "virtually" deleted, or in other words the fact that it has been deleted is recorded in the user isolation scope. A deleted node contains no data, is identified as deleted, and it and all of its descendants "do not exist". In other words, if it is the resource or the ancestor of a resource that would otherwise satisfy a resource request a "resource not found"

error is returned to the requestor. Further details will be outlined in Section 4. In some embodiments, the identification of a node as a deleted node is made by recording the fact in metadata attached to the node, or to the parent of the node, or to some other related entity in the system layer. In other embodiments, a separate repository of deleted node names is maintained, for example, in a separate sub-scope.

3.0 Installation into an isolation environment

The application isolation scope described above can be considered as the scope in which associated application instances share resources independently of any user, or equivalently on behalf of all possible users, including the resources that those application instances create. The main class of such resources is the set created when an application is installed onto the operating system. As shown in Fig. 1A, two incompatible applications cannot both be installed into the same system scope, but this problem can be resolved by installing at least one of those applications into an isolation environment.

An isolation scope, or an application instance associated with an isolation scope, can be operated in an "install mode" to support installation of an application. This is in contrast to "execute mode" described below in connection with FIGs. 4-16. In install mode, the application installation program is associated with an application isolation scope and is presumed to be executing on behalf of all users. The application isolation scope acts, for that application instance, as if it were the user isolation scope for "all users", and no user isolation scope is active for that application instance.

FIG. 3C depicts one embodiment of steps taken in install mode to identify a literal resource when a request to open a native resource is received that indicates the resource is being opened with the intention of modifying it. Briefly, as no user-isolation scope is active, a determination is first made whether the application-scoped instance of the requested native resource exists (step 374). If the application-scoped instance exists, it is identified as the literal resource

instance (step 384). If no application-scoped instance exists, a determination is made whether the system-scoped instance of the requested native resource exists (step 376). If it does not, an error condition is returned to the requestor indicating that the requested virtualized resource does not exist in the virtual scope (step 377). However, if the system-scoped resource exists, it is identified as the candidate resource instance (step 378), and permission data associated with the candidate instance is checked to determine if modification of that instance is allowed (step 380). If not, an error condition is returned to the requestor (step 381) indicating that modification of the virtualized resource is not allowed. If the permission data indicates that the candidate resource may be modified, as no user-isolation scope is active, an application-scoped copy of the candidate instance of the native resource is made (step 382), and the application-scoped instance is identified as the literal instance for the request (step 384). In some embodiments, the candidate file is copied to a location defined by the rules engine. For example, a rule may specify that the file is copied to an application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the file should be copied. Any ancestors of the requested file that do not appear in the isolation scope to which the file is copied are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy.

FIG. 3D shows one embodiment of steps taken in install mode to identify the literal resource when a request to create a native resource is received. Briefly, as no user-isolation scope is active, a determination is first made whether the application-scoped instance of the requested native resource exists (step 390). If the application-scoped instance exists, an error condition may be returned to the requestor indicating that the resource cannot be created because it already exists (step 392). If no application-scoped instance exists, a determination may be made whether the system-scoped instance of the

requested native resource exists (step 394). If the system-scoped instance exists, an error condition may be returned to the requestor indicating that the resource cannot be created because it already exists (step 392). In some embodiments, the request used to open the resource may specify that any extant system-scoped instance of the resource may be overwritten. If the system-scoped resource instance does not exist, the application-scoped resource instance may be identified as the literal instance which will be created to fulfill the request (step 396).

By comparing FIGs. 3B with FIGs. 3C and 3D, it can be seen that install mode operates in a similar manner to execute mode, with the application isolation scope taking the place of the user isolation scope. In other words, modifications to persistent resources, including creation of new resources, take place in the appropriate application isolation scope instead of the appropriate user isolation scope. Furthermore, virtualization of access to existing isolated resources also ignores the appropriate user isolation scope and begins searching for a candidate literal resource in the application isolation scope.

There are two other cases where the application isolation scope operates in this manner to contain modifications to existing resources and creation of new resources. Firstly, there may be an isolation environment configured to operate without a user isolation layer, or a virtual scope configured to operate without a user isolation scope. In this case, the application isolation scope is the only isolation scope that can isolate modified and newly created resources. Secondly, the rules governing a particular set of virtual resources may specify that they are to be isolated into the appropriate application isolation scope rather than into the appropriate user isolation scope. Again, this means modifications to and creations of resources subject to that rule will be isolated into the appropriate application isolation scope where they are visible to all application instances sharing that scope, rather than in the user isolation scope where they are only visible to the user executing those application instances.

In still other embodiments, an isolation environment may be configured to allow certain resources to be shared in the system scope, that is, the isolation environment may act, for one or more system resources, as if no user isolation scope and no application isolation scope exists. System resources shared in the system scope are never copied when accessed with the intent to modify, because they are shared by all applications and all users, i.e., they are global objects.

4.0 Detailed Virtualization Examples

The methods and apparatus described above may be used to virtualize a wide variety of native resources 108. A number of these are described in detail below.

4.1 File System Virtualization

The methods and apparatus described above may be used to virtualize access to a file system. As described above, a file system is commonly organized in a logical hierarchy of directories, which are themselves files and which may contain other directories and data files.

4.1.1 File System Open Operations

In brief overview, FIG. 4 depicts one embodiment of the steps taken to open a file in the virtualized environment described above. A request to open a file is received or intercepted (step 402). The request contains a file name, which is treated as a virtual file name by the isolation environment. The processing rule applicable to the target of the file system open request is determined (step 404). If the rule action is “redirect” (step 406), the virtual file name provided in the request is mapped to a literal file name according to the applicable rule (step 408). A request to open the literal file using the literal file name is passed to the operating system and the result from the operating system is returned to the requestor (step 410). If instead the rule action is “ignore” (step 406), then the literal file name is determined to be exactly the virtual file name (step 412), and the request to open the literal file is passed to the operating system and the

result from the operating system is returned to the requestor (step 410). If in step 406 the rule action is "isolate", then the file name corresponding to the virtual file name in the user isolation scope is identified as the candidate file name (step 414). In other words, the candidate file name is formed by mapping the virtual file name to the corresponding native file name specific to the applicable user isolation scope. The category of existence of the candidate file is determined by examining the user isolation scope and any metadata associated with the candidate file (step 416). If the candidate file is determined to have "negative existence", because either the candidate file or one of its ancestor directories in the user isolation scope is marked as deleted, this means the requested virtual file is known to not exist. In this case, an error condition indicating the requested file is not found is returned to the requestor (step 422). If instead in step 416 the candidate file is determined to have "positive existence", because the candidate file exists in the user isolation scope and is not marked as a placeholder node, then the requested virtual file is known to exist. The candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420). If, however, in step 416 , the candidate file has "neutral existence" because the candidate file does not exist, or the candidate file exists but is marked as a placeholder node, it is not yet known whether the virtual file exists or not. In this case the application-scoped file name corresponding to the virtual file name is identified as the candidate file name (step 424). In other words, the candidate file name is formed by mapping the virtual file name to the corresponding native file name specific to the applicable application isolation scope. The category of existence of the candidate file is determined by examining the application isolation scope and any metadata associated with the candidate file (step 426). If the candidate file is determined to have "negative existence", because either the candidate file or one of its ancestor directories in the application isolation scope is marked as deleted, this means the requested virtual file is known to not exist. In this case, an error

condition indicating the requested file is not found is returned to the requestor (step 422). If instead in step 426 the candidate file is determined to have "positive existence", because the candidate file exists in the application isolation scope and is not marked as a placeholder node, then the requested virtual file is known to exist. The request is checked to determine if the open request indicates an intention to modify the file (step 428). If not, the candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420). If, however, in step 428, it is determined that the open request indicates intention to modify the file, permission data associated with the file is checked to determine if modification of the file is allowed (step 436). If not, an error condition is returned to the requestor (step 438) indicating that modification of the file is not allowed. If the permission data indicates that the file may be modified, the candidate file is copied to the user isolation scope (step 440). In some embodiments, the candidate file is copied to a location defined by the rules engine. For example, a rule may specify that the file is copied to an application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the file should be copied. Any ancestors of the requested file that do not appear in the isolation scope to which the file is copied are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy. The scoped instance is identified as the literal file (step 442) and a request issued to open the literal file and the result returned to the requestor (step 420). Returning to step 426, if the candidate file has neutral existence because the candidate file does not exist, or because the candidate file is found but marked as a placeholder node, it is not yet known whether the virtual file exists or not. In this case, the system-scoped file name corresponding to the virtual file name is identified as the candidate file name (step 430). In other words, the candidate file name is exactly the virtual file name. If the candidate file does not exist (step 432), an error condition indicating

the virtual file was not found is returned to the requestor (step 434). If on the other hand the candidate file exists (step 432), the request is checked to determine if the open request indicates an intention to modify the file (step 428). If not, the candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420). If, however, in step 428, it is determined that the open request indicates intention to modify the file, permission data associated with the file is checked to determine if modification of the file is allowed (step 436). If not, an error condition is returned to the requestor (step 438) indicating that modification of the file is not allowed. If the permission data indicates that the file may be modified, the candidate file is copied to the user isolation scope (step 440). In some embodiments, the candidate file is copied to a location defined by the rules engine. For example, a rule may specify that the file is copied to an application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the file should be copied. Any ancestors of the requested file that do not appear in the isolation scope are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy. The scoped instance is identified as the literal file (step 442) and a request issued to open the literal file and the result returned to the requestor (step 420).

This embodiment can be trivially modified to perform a check for existence of a file rather than opening a file. The attempt to open the literal file in step 420 is replaced with a check for the existence of that literal file and that status returned to the requestor.

Still referring to FIG. 4 and now in more detail, a request to open a virtual file is received or intercepted (step 402). The corresponding literal file may be of user isolation scope, application isolation scope or system scope, or it may be scoped to an application isolation sub-scope or a user isolation sub-scope. In some embodiments, the request is hooked by a function that replaces the

operating system function or functions for opening a file. In another embodiment a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for native files. For embodiments in which a separate operating system function is provided for each type of file operation, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of file operations.

The request contains a file name, which is treated as a virtual file name by the isolation environment. The processing rule applicable to the file system open request is determined (step 404) by consulting the rules engine. In some embodiments the processing rule applicable to the open request is determined using the virtual name included in the open request. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual file name provided for the requested file is used as an index into the rule engine to locate one or more rules that apply to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular file and, in these embodiments, the rule having the longest prefix match with the virtual file name is the rule applied to the request. In other embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 4 as a single database transaction or single lookup into a file, the rule lookup may be performed as a series of rule lookups.

If the rule action is "redirect" (step 406), the virtual file name provided in the request is mapped to a literal file name according to the applicable rule (step 408). A request to open the literal file identified by the literal file name is passed to the operating system and the result from the operating system is returned to the requestor (step 410). For example, a request to open a file named "file_1" may result in the opening of a literal file named "Different_ file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to open the file using the virtual name results in the return of a STATUS_REPARSE response from the file system filter driver indicating the determined literal name. The I/O Manager then reissues the file open request with the determined literal name include in the STATUS_REPARSE response.

If instead the rule action is "ignore" (step 406), then the literal file name is determined to be exactly the virtual file name (step 412), and the request to open the literal file is passed to the operating system and the result from the operating system is returned to the requestor (step 410). For example, a request to open a file named "file_1" will result in the opening of an actual file named "file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument.

If in step 406 the rule action is "isolate", then the user-scoped file name corresponding to the virtual file name is identified as the candidate file name (step 414). In other words, the candidate file name is formed by mapping the virtual file name to the corresponding native file name specific to the applicable user isolation scope. For example, a request to open a file named "file_1" may result in the opening of an actual file named "Isolated_file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For

embodiments using a file system filter driver, the first request to open the file using the virtual name results in the return of a STATUS_REPARSE response from the file system filter driver indicating the determined literal name. The I/O Manager then reissues the file open request with the determined literal name include in the REPARSE response.

In some embodiments, the literal name formed in order to isolate a requested system file may be based on the virtual file name received and a scope-specific identifier. The scope-specific identifier may be an identifier associated with an application isolation scope, a user isolation scope, a session isolation scope, an application isolation sub-scope, a user isolation sub-scope, or some combination of the above. The scope-specific identifier is used to "mangle" the virtual name received in the request.

In other embodiments, the user isolation scope or a sub-scope may be a directory under which all files that exist in the user isolation scope are stored. In some of these embodiments, the directory tree structure under the user isolation directory reflects the path of the requested resource. In other words, the literal file path is formed by mapping the virtual file path to the user isolation scope. For example, if the requested file is c:\temp\test.txt and the user isolation scope directory is d:\user1\app1\, then the path to the user-scoped literal file may be d:\user1\app1\c\temp\test.txt. In other embodiments, the path to the user-scoped literal may be defined in a native naming convention. For example, the path to the user-scoped literal file may be d:\user1\app1\device\harddisk1\temp\test.txt. In still other embodiments, the user-scoped files may all be stored in a single directory with names chosen to be unique and a database may be used to store the mapping between the requested file name and the name of the corresponding literal file stored in the directory. In still other embodiments, the contents of the literal files may be stored in a database. In still other embodiments, the native file system provides the facility for a single file to contain multiple independent named "streams", and

the contents of the user-scoped files are stored as additional streams of the associated files in the system scope. Alternatively, the literal files may be stored in a custom file system that may be designed to optimize disk usage or other criteria of interest.

The category of existence of the candidate file is determined by examining the user isolation scope and any metadata associated with the candidate file (step 416). If the candidate file is determined to have "negative existence", because either the candidate file or one of its ancestor directories in the user isolation scope is marked as deleted, this means the requested virtual file is known to not exist. In this case, an error condition indicating the requested file is not found is returned to the requestor (step 422).

In some embodiments, small amounts of metadata about a file may be stored directly in the literal filename, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the file by virtual filename check for possible variations of the literal filename due to the presence of a metadata indicator, and requests to retrieve the name of the file itself are hooked or intercepted in order to respond with the literal name. In other embodiments, one or more alternate names for the file may be formed from the virtual file name and a metadata indicator, and may be created using hard link or soft link facilities provided by the file system. The existence of these links may be hidden from applications by the isolation environment by indicating that the file is not found if a request is given to access a file using the name of a link. A particular link's presence or absence may indicate one bit of metadata for each metadata indicator, or there may be a link with a metadata indicator that can take on multiple states to indicate several bits of metadata. In still other embodiments, where the file system supports alternate file streams, an alternate file stream may be created to embody metadata, with the size of the stream indicating several

bits of metadata. In still other embodiments, a file system may directly provide the ability to store some 3rd party metadata for each file in the file system.

In specific ones of these embodiments, a list of deleted files or file system elements may be maintained and consulted to optimize this check for deleted files. In these embodiments, if a deleted file is recreated then the file name may be removed from the list of deleted files. In others of these embodiments, a file name may be removed from the list if the list grows beyond a certain size.

If instead in step 416 the candidate file is determined to have "positive existence", because the candidate file exists in the user isolation scope and is not marked as a placeholder node, then the requested virtual file is known to exist. The candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420).

If, however, in step 416, the candidate file has "neutral existence" because the candidate file does not exist, or the candidate file exists but is marked as a placeholder node, it is not yet known whether the virtual file exists or not. In this case the application-scoped file name corresponding to the virtual file name is identified as the candidate file name (step 424). In other words, the candidate file name is formed by mapping the virtual file name to the corresponding native file name specific to the applicable application isolation scope. The category of existence of the candidate file is determined by examining the application isolation scope and any metadata associated with the candidate file (step 426).

If the application-scoped candidate file is determined to have "negative existence", because either the candidate file or one of its ancestor directories in the application isolation scope is marked as deleted, this means the requested virtual file is known to not exist. In this case, an error condition indicating the requested file is not found is returned to the requestor (step 422).

If in step 426 the candidate file is determined to have "positive existence", because the candidate file exists in the application isolation scope and is not

marked as a placeholder node, then the requested virtual file is known to exist. The request is checked to determine if the open request indicates an intention to modify the file (step 428). If not, the candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420).

If, however, in step 428, it is determined that the open request indicates intention to modify the file, permission data associated with the file is checked to determine if modification of the file is allowed (step 436). In some embodiments, the permission data is associated with the application-scoped candidate file. In some of these embodiments, the permissions data is stored in a rules engine or in metadata associated with the candidate file. In other embodiments, the permission data associated with the candidate file is provided by the operating system. Further, the rules engine may include configuration settings instructing the isolation environment to obey or override the native permission data for virtualized copies of resources. In some embodiments, the rules may specify for some virtual resources the scope in which modifications are to occur, for example the system scope or the application isolation scope or a sub-scope, or the user isolation scope or a sub-scope. In some embodiments, the rules engine may specify configuration settings that apply to subsets of the virtualized native resources based on hierarchy or on type of resource accessed. In some of these embodiments, the configuration settings may be specific to each atomic native resource. In another example, the rules engine may include configuration data that prohibits or allows modification of certain classes of files, such as executable code or MIME types or file types as defined by the operating system.

If the permission data associated with the candidate file indicates that it may not be modified, an error condition is returned to the requestor (step 438) indicating that modification of the file is not allowed. If the permission data indicates that the file may be modified, the candidate file is copied to the user isolation scope (step 440). In some embodiments, the candidate file is copied to

a location defined by the rules engine. For example, a rule may specify that the file is copied to another application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the file should be copied. Any ancestors of the requested file that do not appear in the isolation scope to which the file is copied are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy.

In some embodiments, metadata is associated with files copied to the isolation scope that identifies the date and time at which the files were copied. This information may be used to compare the time stamp associated with the copied instance of the file to the time stamp of the last modification of the original instance of the file or of another instance of the file located in a lower isolation scope. In these embodiments, if the original instance of the file, or an instance of the file located in a lower isolation scope, is associated with a time stamp that is later than the time stamp of the copied file, that file may be copied to the isolation scope to update the candidate file. In other embodiments, the copy of the file in the isolation scope may be associated with metadata identifying the scope containing the original file that was copied.

In further embodiments, files that are copied to isolation scopes because they have been opened with intent to modify them may be monitored to determine if they are, in fact, modified. In one embodiment a copied file may be associated with a flag that is set when the file is actually modified. In these embodiments, if a copied file is not actually modified, it may be removed from the scope to which it was copied after it is closed, as well as any placeholder nodes associated with the copied file.

The scoped instance is identified as the literal file (step 442) and a request issued to open the literal file and the result returned to the requestor (step 420).

Returning to step 426, if the candidate file has neutral existence because the candidate file does not exist, or if the candidate file is found but marked as a

placeholder node, it is not yet known whether the virtual file exists or not. In this case, the system-scoped file name corresponding to the virtual file name is identified as the candidate file name (step 430). In other words, the candidate file name is exactly the virtual file name.

If the candidate file does not exist (step 432), an error condition indicating the virtual file was not found is returned to the requestor (step 434). If on the other hand the candidate file exists (step 432), the request is checked to determine if the open request indicates an intention to modify the file (step 428).

As above, if the candidate file is being opened without the intent to modify it, the system-scoped candidate file is identified as the literal file for the request (step 418), and a request issued to open the literal file and the result returned to the requestor (step 420). If, however, in step 428, it is determined that the open request indicates intention to modify the file, permission data associated with the file is checked to determine if modification of the file is allowed (step 436). In some embodiments, the permission data is associated with the system-scoped candidate file. In some of these embodiments, the permissions data is stored in a rules engine or in metadata associated with the candidate file. In other embodiments, the permission data associated with the candidate file is provided by the operating system.

If the permission data associated with the system-scoped candidate file indicates that the file may not be modified, an error condition is returned to the requestor (step 438) indicating that modification of the file is not allowed. If, however, the permission data indicates that the file may be modified, the candidate file is copied to the user isolation scope (step 440). In some embodiments, the candidate file is copied to a location defined by the rules engine. For example, a rule may specify that the file is copied to an application isolation scope or that it may be left in the system scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the file should be copied. Any ancestors of the requested file

that do not appear in the isolation scope are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy.

In some embodiments, metadata is associated with files copied to the isolation scope that identifies the date and time at which the files were copied. This information may be used to compare the time stamp associated with the copied instance of the file to the time stamp of the last modification of the original instance of the file. In these embodiments, if the original instance of the file is associated with a time stamp that is later than the time stamp of the copied file, the original file may be copied to the isolation scope to update the candidate file. In other embodiments, the candidate file copied to the isolation scope may be associated with metadata identifying the scope from which the original file was copied.

In further embodiments, files that are copied to isolation scopes because they have been opened with intent to modify them may be monitored to determine if they are, in fact, modified. In one embodiment a copied file may be associated with a flag that is set when the file is actually modified. In these embodiments, if a copied file is not actually modified, when it is closed it may be removed from the scope to which it was copied, as well as any placeholder nodes associated with the copied file. In still further embodiments, the file is only copied to the appropriate isolation scope when the file is actually modified.

The scoped instance is identified as the literal file (step 442) and a request issued to open the literal file and the result returned to the requestor (step 420).

4.1.2 File System Delete Operations

Referring now to FIG. 5, and in brief overview, one embodiment of the steps taken to delete a file is depicted. A request to delete a file is received or intercepted (step 502). The request contains a file name, which is treated as a virtual file name by the isolation environment. A rule determines how the file operation is processed (step 504). If the rule action is “redirect” (step 506), the virtual file name is mapped directly to a literal file name according to the rule

(step 508). A request to delete the literal file is passed to the operating system and the result from the operating system is returned to the requestor (step 510). If the rule action is "ignore" (step 506), then the literal file name is identified as exactly the virtual file name (step 513), and a request to delete the literal file is passed to the operating system and the result from the operating system is returned to the requestor (step 510). If the rule action is "isolate" (step 506), then the existence of the virtual file is determined (step 514). If the virtual file does not exist, an error condition is returned to the requestor indicating that the virtual file does not exist (step 516). If the virtual file exists, and if the virtualized file specifies a directory rather than an ordinary file, the virtual directory is consulted to determine if it contains any virtual files or virtual subdirectories (step 518). If the requested virtualized file is a virtual directory that contains any virtual files or virtual subdirectories, the virtual directory cannot be deleted and an error message is returned (step 520). If the requested virtualized file is an ordinary file or is a virtual directory that contains no virtual files and no virtual subdirectories, then the literal file corresponding to the virtual file is identified (step 522). Permission data associated with the file is checked to determine if deletion is allowed (step 524). If not, a permission error message is returned (step 526). If, however, deletion of the file is allowed, and if the literal file is in the appropriate user isolation scope (step 528), the literal file is deleted (step 534) and a "deleted" node representing the deleted virtual file is created in the appropriate user isolation scope (step 536). If, however, in step 528 it is determined that the literal file is not in the user isolation scope but is in the appropriate application isolation scope or the system scope, then an instance of every user-scoped ancestor of the user-scoped instance of the requested file that does not already exist is created and marked as a placeholder (step 532). This is done to maintain the logical hierarchy of the directory structure in the user isolation scope. A user-scoped "deleted" node representing the deleted virtual file is then created in the appropriate user isolation scope (step 536).

Still referring to FIG. 5, and in more detail, a request to delete a file is received or intercepted (step 502). The file may be of user isolation scope, application isolation scope, system scope, or some applicable isolation sub-scope. In some embodiments, the request is hooked by a function that replaces the operating system function or functions for deleting the file. In another embodiment a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for native files. For embodiments in which a separate operating system function is provided for each type of file, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of files.

The request contains a file name, which is treated as a virtual file name by the isolation environment. A processing rule applicable to the delete operation is determined (step 504) by consulting the rules engine. In some embodiments, the virtual file name provided for the requested file is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular file and, in these embodiments, the rule having the longest prefix match with the virtual file name is the rule applied to the request. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual file name provided in the request is used as an index into a rules engine to locate one or more rules that apply to the request. In other embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be

to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 5 as a series of decisions, the rule lookup may occur as a single database transaction.

If the rule action is "redirect" (step 506), the virtual file name is mapped directly to a literal file name according to the applicable rule (step 508). A request to delete the literal file is passed to the operating system and the result from the operating system is returned to the requestor (step 510). For example, a request to delete a file named "file_1" may result in the deletion of an actual file named "Different_ file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to delete the file using the virtual name results in the return of a STATUS_REPARSE response from the file system filter driver indicating the determined literal name. The I/O Manager then reissues the file delete request with the determined literal name include in the STATUS_REPARSE response.

In some embodiments, operating system permissions associated with the literal file "Different_file_1" may prevent deletion of the literal file. In these embodiments, an error message is returned that the file could not be deleted.

If the rule action is "ignore" (step 506), then the literal file name is identified as exactly the virtual file name (step 513), and a request to delete the literal file is passed to the operating system and the result from the operating system is returned to the requestor (step 510). For example, a request to delete a file named "file_1" will result in the deletion of an actual file named "file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to delete the file using the virtual name results in the return of a STATUS_REPARSE response from the file system filter driver indicating the

literal name. The I/O Manager then reissues the file delete request with the determined literal name include in the STATUS_REPARSE response.

In some embodiments, operating system permissions associated with the literal file "file_1" may prevent deletion of the literal file. In these embodiments, an error message is returned that the file could not be deleted.

If the rule action is "isolate" (step 506), then the existence of the virtual file is determined (step 514). If the file does not exist, an error is returned indicating that the file is not found (step 516).

If, however, in step 518 it is determined that the file exists but that it is not an ordinary file and is not an empty virtual directory, i.e., it contains virtual files or virtual subdirectories, an error message is returned indicating that the file may not be deleted (step 520).

If, however, the file is determined to exist and the requested virtualized file is an ordinary file or is an empty virtual directory, i.e., it contains no virtual files and no virtual subdirectories (step 518), then the literal file corresponding to the virtual file is identified (step 522). The literal file name is determined from the virtual file name as specified by the isolation rule. For example, a request to delete a file named "file_1" may result in the deletion of an actual file named "Isolated_file_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to delete the file using the virtual name results in the return of a STATUS_REPARSE response from the file system filter driver indicating the literal name. The I/O Manager then reissues the file delete request with the determined literal name include in the STATUS_REPARSE response.

Once the literal file corresponding the virtual file is identified, it is determined whether the literal file may be deleted (step 524). If the file may not be deleted, an error is returned indicating that the file could not be deleted (step 524). In some embodiments, the permission data is associated with the system-

scoped candidate file. In some of these embodiments, the permissions data is stored in a rules engine or in metadata associated with the candidate file. In other embodiments, the permission data associated with the candidate file is provided by the operating system.

If, however, deletion of the file is allowed, and if the literal file is in the appropriate user isolation scope (step 528), the literal file is deleted (step 534) and a "deleted" node representing the deleted virtual file is created in the appropriate user isolation scope (step 536).

If, however, in step 528 it is determined that the literal file is not in the user isolation scope but is in the appropriate application isolation scope or the system scope, then an instance of every user-scoped ancestor of the user-scoped instance of the requested file that does not already exist is created and marked as a placeholder (step 532). This is done to maintain the logical hierarchy of the directory structure in the user isolation scope. A user-scoped "deleted" node representing the deleted virtual file is then created in the appropriate user isolation scope (step 536). In some embodiments, the identity of the deleted file is stored in a file or other cache memory to optimize checks for deleted files.

In some embodiments, the located virtualized file may be associated with metadata indicating that the virtualized file has already been deleted. In some other embodiments, an ancestor of the virtualized file (e.g., a higher directory containing the file) is associated with metadata indicating that it is deleted. In these embodiments, an error message may be returned indicating that the virtualized file does not exist. In specific ones of these embodiments, a list of deleted files or file system elements may be maintained and consulted to optimize this check for deleted files.

4.1.3 File System Enumeration Operations

Referring now to FIG. 6, and in brief overview, one embodiment of the steps taken to enumerate a directory in the described virtualized environment is shown. A request to enumerate is received or intercepted (step 602). The

request contains a directory name that is treated as a virtual directory name by the isolation environment. Conceptually, the virtual directory's existence is determined as described in section 4.1.1 (step 603). If the virtual directory does not exist, a result indicating that the virtual directory is not found is returned to the requestor (step 620). If instead the virtual directory exists, the rules engine is consulted to determine the rule for the directory specified in the enumerate request (step 604). If the rule specifies an action of "redirect" (step 606), the literal directory name corresponding to the virtual directory name is determined as specified by the rule (step 608) and the literal directory identified by the literal name is enumerated, and the enumeration results stored in a working data store (step 612), followed by step 630 as described later. If the rule action specified is not "redirect" and is "ignore," (step 610) the literal directory name is exactly the virtual directory name (step 613) and the literal directory is enumerated, and the enumeration results stored in a working data store (step 612), followed by step 630 as described later. If, however, the rule action specifies "isolate," firstly the system scope is enumerated; that is, the candidate directory name is exactly the virtual directory name, and if the candidate directory exists it is enumerated. The enumeration results are stored in a working data store. If the candidate directory does not exist, the working data store remains empty at this stage (step 614). Next, the candidate directory is identified as the application-scoped instance of the virtual directory, and the category of existence of the candidate directory is determined (step 615). If the candidate directory has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 642). If instead the candidate directory does not have negative existence, the candidate directory is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each file system element in the enumeration, its category of existence is determined. Elements with negative existence are removed from the working data store, and elements with positive

existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding element if one is already present in the working data store (step 616).

In either case, the candidate directory is identified as the user-scoped instance of the virtual directory, and the category of existence of the candidate directory is determined (step 617). If the candidate directory has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 644). If instead the candidate directory does not have negative existence, the candidate directory is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each file system element in the enumeration, its category of existence is determined. Elements with negative existence are removed from the working data store, and elements with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding element if one is already present in the working data store (step 618), followed by step 630 as described below.

Then, for all three types of rules, step 630 is executed. The rules engine is queried to find the set of rules whose filters match immediate children of the requested directory, but do not match the requested directory itself (step 630). For each rule in the set, the existence of the virtual child whose name matches the name in the rule is queried using the logic outlined in section 4.1.1. If the child has positive existence, it is added to the working data store, replacing any child of the same name already there. If the child has negative existence, the entry in the working data store corresponding to the child, if any, is removed. (Step 632). Finally, the constructed enumeration is then returned from the working data store to the requestor (step 620).

Still referring to FIG. 6, and in more detail, a request to enumerate a directory is received or intercepted (step 602). In some embodiments, the request is hooked by a function that replaces the operating system function or functions for enumerating a directory. In another embodiment, a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for file operations. For embodiments in which a separate operating system function is provided for each type of file operation, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of file operations.

The existence of the virtual directory is determined (step 603). This is achieved as described in section 4.1.1. If the virtual directory does not exist, it cannot be enumerated, and a result indicating that the virtual directory does not exist is returned to the requestor (step 620).

The request contains a directory name, which is treated as a virtual directory name by the isolation environment. If the virtual directory exists, then a rule determining how the enumeration operation is to be processed is located (step 604) by consulting the rules engine. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual directory name provided for the requested directory is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular directory and, in these embodiments, the rule having the longest prefix match with the virtual directory name is the rule applied to the

request. In other embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 6 as a single database transaction or single lookup into a file, the rule lookup may be performed as a series of rule lookups.

If the rule action is "redirect" (step 606), the virtual directory name is mapped directly to a literal directory name according to the rule (step 608). A request to enumerate the literal directory is passed to the operating system (step 612) and step 630 is executed as described later. For example, a request to enumerate a directory named "directory _1" may result in the enumeration of a literal directory named "Different_ directory_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to open the directory for enumeration using the virtual name results in a "STATUS_REPARSE" request response indicating the determined literal name. The I/O Manager then reissues the directory open request for enumeration with the determined literal name include in the STATUS_REPARSE response.

If the rule action is not "redirect" (step 606), but is "ignore" (step 610), then the literal directory name is identified as exactly the virtual directory name (step 613), and a request to enumerate the literal directory is passed to the operating system (step 612) and step 630 is executed as described later. For example, a request to enumerate a directory named "directory _1" will result in the enumeration of an actual directory named "directory _1." In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. For embodiments using a file system filter driver, the first request to enumerate the directory using the virtual name is passed on unmodified by the filter driver.

If the rule action determined in step 610 is not "ignore" but is "isolate", then the system scope is enumerated, that is, the virtual name provided in the request is used to identify the enumerated directory (step 614). The results of the enumeration are stored in a working data store. In some embodiments, the working data store is comprised of a memory element. In other embodiments, the working data store comprises a database or a file or a solid-state memory element or a persistent data store.

Next, the candidate directory is identified as the application-scoped instance of the virtual directory, and the category of existence of the candidate directory is determined (step 615). If the candidate directory has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 642).

In some embodiments, small amounts of metadata about a file may be stored directly in the literal filename, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the file by virtual filename check for possible variations of the literal filename due to the presence of a metadata indicator, and requests to retrieve the name of the file itself are hooked or intercepted in order to respond with the literal name. In other embodiments, one or more alternate names for the file may be formed from the virtual file name and a metadata indicator, and may be created using hard link or soft link facilities provided by the file system. The existence of these links may be hidden from applications by the isolation environment by indicating that the file is not found if a request is given to access a file using the name of a link. A particular link's presence or absence may indicate one bit of metadata for each metadata indicator, or there may be a link with a metadata indicator that can take on multiple states to indicate several bits of metadata. In still other embodiments,

where the file system supports alternate file streams, an alternate file stream may be created to embody metadata, with the size of the stream indicating several bits of metadata. In still other embodiments, a file system may directly provide the ability to store some 3rd party metadata for each file in the file system. In yet other embodiment, a separate sub-scope may be used to record deleted files, and existence of a file (not marked as a placeholder) in that sub-scope is taken to mean that the file is deleted.

If instead the candidate directory does not have negative existence, the candidate directory is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each file system element in the enumeration, its category of existence is determined. Elements with negative existence are removed from the working data store, and elements with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding element if one is already present in the working data store (step 616).

In either case, the candidate directory is identified as the user-scoped instance of the virtual directory, and the category of existence of the candidate directory is determined (step 617). If the candidate directory has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 644). If instead the candidate directory does not have negative existence, the candidate directory is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each file system element in the enumeration, its category of existence is determined. Elements with negative existence are removed from the working data store, and elements with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data

store, replacing the corresponding element if one is already present in the working data store (step 618), followed by step 630 as described below.

Then, for all three types of rules, step 630 is executed. The rules engine is queried to find the set of rules whose filters match immediate children of the requested directory, but do not match the requested directory itself (step 630). For each rule in the set, the existence of the virtual child whose name matches the name in the rule is queried using the logic outlined in section 4.1.1. If the child has positive existence, it is added to the working data store, replacing any child of the same name already there. If the child has negative existence, the entry in the working data store corresponding to the child, if any, is removed. (Step 632). Finally, the constructed enumeration is then returned from the working data store to the requestor (step 620).

A practitioner of ordinary skill in the art will realize that the layered enumeration process described above can be applied with minor modification to the operation of enumerating a single isolation scope which comprises a plurality of isolation sub-scopes. A working data store is created, successive sub-scopes are enumerated and the results are merged into the working data store to form the aggregated enumeration of the isolation scope.

4.1.4. File System Creation Operations

Referring now to FIG. 7, and in brief overview, one embodiment of the steps taken to create a file in the isolation environment is shown. A request to create a file is received or intercepted (step 702). The request contains a file name, which is treated as a virtual file name by the isolation environment. An attempt is made to open the requested file using full virtualization using applicable rules, i.e. using appropriate user and application isolation scope, as described in section 4.1.1 (step 704). If access is denied (step 706), an access denied error is returned to the requestor (step 709). If access is granted (step 706), and the requested file is successfully opened (step 710), the requested file is returned to the requestor (step 712). However, if access is granted (step 706),

but the requested file is not opened successfully (step 710) then if the parent of the requested file also does not exist (step 714), an error appropriate to the request semantics is issued to the requestor (step 716). If on the other hand, the parent of the requested file is found in full virtualized view using the appropriate user and application scope (step 714), a rule then determines how the file operation is processed (step 718). If the rule action is “redirect” or “ignore” (step 720), the virtual file name is mapped directly to a literal file name according to the rule. Specifically, if the rule action is “ignore”, the literal file name is identified as exactly the virtual file name. If, instead, the rule action is “redirect”, the literal file name is determined from the virtual file name as specified by the rule. Then a request to create the literal file is passed to the operating system, and the result is returned to the requestor (step 724). If on the other hand, the rule action determined in step 720 is “isolate”, then the literal file name is identified as the instance of the virtual file name in the user isolation scope. If the literal file already exists, but is associated with metadata indicating that it is a placeholder or that it is deleted, then the associated metadata is modified to remove those indications, and it is ensured that the file is empty. In either case, a request to open the literal file is passed to the operating system (step 726). If the literal file was opened successfully (step 728), the literal file is returned to the requestor (step 730). If on the other hand, in step 728, the requested file fails to open, placeholders for each ancestor of the literal file that does not currently exist in the user-isolation scope (step 732) and a request to create the literal file using the literal name is passed to the operating system and the result is returned to the requestor (step 734).

Still referring to FIG. 7, and in more detail, a request to create a file is received or intercepted (step 702). In some embodiments, the request is hooked by a function that replaces the operating system function or functions for creating the file. In another embodiment, a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in

kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for files. For embodiments in which a separate operating system function is provided for each type of file operation, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of file operations.

The request contains a file name, which is treated as a virtual file name by the isolation environment. The requestor attempts to open the requested file using full virtualization using applicable rules, i.e. using appropriate user and application isolation scope, as described in section 4.1.1 (step 704). If access is denied during the full virtualized open operation (step 706), an access denied error is returned to the requestor (step 709). If access is granted (step 706), and the requested virtual file is successfully opened (step 710), the corresponding literal file is returned to the requestor (step 712). However, if access is granted (step 706), but the requested file is not opened successfully (step 710) then the virtual file has been determined not to exist. If the virtual parent of the requested virtual file also does not exist, as determined by the procedures in section 4.1.1 (step 714), an error appropriate to the request semantics is issued to the requestor (step 716). If on the other hand, the virtual parent of the requested virtual file is found in full virtualized view using the appropriate user and application scope (step 714), then a rule that determines how the create operation is processed is located (step 718) by consulting the rules engine. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual file name provided for the requested file is used to locate in the rule engine a rule that

applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular file and, in some of these embodiments, the rule having the longest prefix match with the virtual file name is the rule applied to the request. In some embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 7 as a single database transaction or single lookup into a file, the rule lookup may be performed as a series of rule lookups..

If the rule action is “redirect” or “ignore” (step 720), the virtual file name is mapped directly to a literal file name according to the rule (step 724). If the rule action is “redirect” (step 720), the literal file name is determined from the virtual file name as specified by the rule (step 724). If the rule action is “ignore” (step 720), the literal file name is determined to be exactly the virtual file name (step 724). If the rule action is “ignore” or the rule action is “redirect”, a request to create the literal file using the determined literal file name is passed to the operating system and the result from the operating system is returned to the requestor (step 724). For example, a request to create a virtual file named “file_1” may result in the creation of a literal file named "Different_ file_1.” In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. (step 724). For embodiments using a file system filter driver, the first request to open the file using the virtual name results in a “STATUS_REPARSE” request response that indicates the determined literal name. The I/O Manager then reissues the file open request with the determined literal name include in the STATUS_REPARSE response.

If the rule action determined in step 720 is not “ignore” or “redirect” but is “isolate,” then the literal file name is identified as the instance of the virtual file name in the user isolation scope. If the literal file already exists, but is

associated with metadata indicating that it is a placeholder or that it is deleted, then the associated metadata is modified to remove those indications, and it is ensured that the file is empty.

In some embodiments, small amounts of metadata about a file may be stored directly in the literal filename, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the file by virtual filename check for possible variations of the literal filename due to the presence of a metadata indicator, and requests to retrieve the name of the file itself are hooked or intercepted in order to respond with the literal name. In other embodiments, one or more alternate names for the file may be formed from the virtual file name and a metadata indicator, and may be created using hard link or soft link facilities provided by the file system. The existence of these links may be hidden from applications by the isolation environment by indicating that the file is not found if a request is given to access a file using the name of a link. A particular link's presence or absence may indicate one bit of metadata for each metadata indicator, or there may be a link with a metadata indicator that can take on multiple states to indicate several bits of metadata. In still other embodiments, where the file system supports alternate file streams, an alternate file stream may be created to embody metadata, with the size of the stream indicating several bits of metadata. In still other embodiments, a file system may directly provide the ability to store some 3rd party metadata for each file in the file system.

In specific ones of these embodiments, a list of deleted files or file system elements may be maintained and consulted to optimize this check for deleted files. In these embodiments, if a deleted file is recreated then the file name may be removed from the list of deleted files. In others of these embodiments, a file name may be removed from the list if the list grows beyond a certain size.

In either case, a request to open the user-scoped literal file is passed to the operating system (step 726). In some embodiments, rules may specify that the literal file corresponding to the virtual file should be created in a scope other than the user isolation scope, such as the application isolation scope, the system scope, a user isolation sub-scope or an application isolation sub-scope.

If the literal file was opened successfully (step 728), the literal file is returned to the requestor (step 730). If on the other hand, in step 728, the requested file fails to open, placeholders are created for each ancestor of the literal file that does not currently exist in the user-isolation scope (step 732) and a request to create the literal file using the literal name is passed to the operating system and the result is returned to the requestor (step 734).

This embodiment is for operating systems with APIs or facilities that only support creation of one level per call/invocation. Extension to multi-levels per call/invocation should be obvious to one skilled in the art.

4.1.5 Short filename management

In some file systems, both short and long filenames may be given to each file. Either name may be used to access the file in any of the file operations described above. For each file that possesses both a short and long filename, this implicitly creates an association between the short and long filename assigned to that file. In some of these file systems, short names are automatically assigned by the file system to files that are created using long file names. If the association between short and long filename is not maintained by the isolation environment, files with different long names in the same directory but in different scope levels may have the same short file name, leading to ambiguity if the short name is used to access a virtual file. Alternately, the short file name may change when a file is copied to a user isolation scope for modification meaning the virtual file can no longer be accessed using the original short name.

In order to prevent these issues, firstly file system operations that copy file instances opened with intention to modify to a "higher" scope preserve the association between the short and long filenames associated with the copied instance. Secondly, unique short names are created for newly-created isolated files in lieu of the filenames assigned by the operating system. The generated short filenames should satisfy the condition that the generated filenames do not match any existing short filenames in the same directory in the same isolation scope or in the same directory in a "lower" isolation scope. For example, a short filename generated for an instance of a file located in a user isolation scope should not match existing short filenames in application-scoped instance of the directory or in the system-scoped instance of the directory.

Referring now to FIG. 7A, one embodiment of the steps taken to assign unique short filenames after creating a new file is shown. In brief overview, a check is made to determine if short filenames should be generated (step 752). If not, a status is returned indicating that no short filename will be generated (step 754). Otherwise, the filename is checked to determine if it is already a legal short filename according to the file system (step 756). If it is already a legal short filename, a status is returned indicating that no short name will be generated (step 754). Otherwise, a suitable short filename is constructed (step 758).

Still referring to FIG. 7A, and in greater detail, a check is made to determine if short filenames should be generated (step 752). In some embodiments, this decision may be made based on the device storing the file to which the filename refers. In other embodiments, generation of short filenames may be enabled for certain scopes or sub-scopes, or for the isolation environment as a whole. In some of these embodiments, a registry setting may specify whether a short filename will be generated for a particular filename. If no short filename should be generated, a status that no short filename will be generated is returned (step 754).

Otherwise, the filename is checked to determine if it is already a legal short filename (step 756). In some embodiments, legal short filenames contain up to eight characters in the filename and up to three characters in an optional extension. In some embodiments, legal short names contain only legal characters, such as A-Z, a-z, 0-9, `, ~, !, @, #, $, %, ^, &, *, (,), -, _, ‘, {, and }. In some embodiments a leading space or “.” or more than one embedded “.” is illegal. If the provided filename is already a legal short filename, a status is returned that no short filename will be generated (step 754).

Otherwise, if it is determined in step 756 that the filename is an illegal short filename, a suitable short filename is constructed (step 758). In some embodiments this is achieved by using some of the parts of the long filename that are legal to use in a short filename, combined with an encoded iteration count to form a candidate short filename. The iteration count is increased until the associated candidate short filename is suitable, that is it is a legal short filename that is not used by any other file in the same directory in the same scope, or in the same directory in a lower scope. In other embodiments, the long filename is mangled or hashed and encoded, and is combined with an encoded iteration count to form a candidate short filename. The iteration count is increased until the associated candidate short filename is suitable, that is it is a legal short filename that is not used by any other file in the same directory in the same scope, or in the same directory in a lower scope. In all of these embodiments a scope-specific string may be incorporated into the candidate short filename to increase the likelihood that a suitable candidate short filename will be found with a low iteration count.

4.2 Registry Virtualization

The methods and apparatus described above may be used to virtualize access to a registry database. As described above a registry database stores information regarding hardware physically attached to the computer, which system options have been selected, how computer memory is set up, various

items of application-specific data, and what application programs should be present when the operating system is started. A registry database is commonly organized in a logical hierarchy of “keys” 170, 172, which are containers for registry values.

4.2.1 Registry Key Open Operations

In brief overview, FIG. 8 depicts one embodiment of the steps taken to open a registry key in the isolation environment described above. A request to open a registry key is received or intercepted, the request containing a registry key name which is treated as a virtual key name by the isolation environment (step 802). A processing rule applicable to the virtual name in the request determines how the registry key operation is processed (step 804). If the rule action is “redirect” (step 806), the virtual key name provided in the request is mapped to a literal key name as specified by the applicable rule (step 808). A request to open the literal registry key using the literal key name is passed to the operating system and the result from the operating system is returned to the requestor (step 810). If the rule action is not “redirect”, but is “ignore” (step 806), then the virtual key name is identified as the literal key name (step 812), and a request to open the literal registry key is passed to the operating system and the result from the operating system is returned to the requestor (step 810). If the rule action determined in step 806 is not “redirect” and is not “ignore,” but is “isolate”, the virtual key name provided in the request is mapped to a user-scoped candidate key name, that is a key name corresponding to the virtual key name that is specific to the applicable user isolation scope (step 814). The category of existence of the user-scoped candidate key is determined by examining the user isolation scope and any metadata associated with the candidate key (step 816). If the candidate key is determined to have “negative existence”, because either the candidate key or one of its ancestor keys in the user isolation scope is marked as deleted, this means the requested virtual key is known to not exist. In this case, an error condition indicating the requested file is

not found is returned to the requestor (step 822). If instead in step 816 the candidate key is determined to have “positive existence”, because the candidate key exists in the user isolation scope and is not marked as a placeholder node, then the requested virtual key is known to exist. The candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820). If, however, in step 816, the candidate key has “neutral existence” because the candidate key does not exist, or the candidate key exists but is marked as a placeholder node, it is not yet known whether the virtual key exists or not. In this case the application-scoped key name corresponding to the virtual key name is identified as the candidate key name (step 824). In other words, the candidate key name is formed by mapping the virtual key name to the corresponding native key name specific to the applicable application isolation scope. The category of existence of the candidate key is determined by examining the application isolation scope and any metadata associated with the candidate key (step 826). If the candidate key is determined to have “negative existence”, because either the candidate key or one of its ancestor keys in the application isolation scope is marked as deleted, this means the requested virtual key is known to not exist. In this case, an error condition indicating the requested key is not found is returned to the requestor (step 822). If instead in step 826 the candidate key is determined to have “positive existence”, because the candidate key exists in the application isolation scope and is not marked as a placeholder node, then the requested virtual key is known to exist. The request is checked to determine if the open request indicates an intention to modify the key (step 828). If not, the candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820). If, however, in step 828, it is determined that the open request indicates an intention to modify the key, permission data associated with the key is checked to determine if modification of the key is allowed (step 836). If not, an error

condition is returned to the requestor (step 838) indicating that modification of the key is not allowed. If the permission data indicates that the key may be modified, the candidate key is copied to the user isolation scope (step 840). In some embodiments, the candidate key is copied to a location defined by the rules engine. For example, a rule may specify that the key is copied to an application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the key should be copied. Any ancestors of the requested key that do not appear in the isolation scope to which the key is copied are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy. The newly copied scoped instance is identified as the literal key (step 842) and a request issued to open the literal key and the result returned to the requestor (step 820). Returning to step 826, if the candidate key has neutral existence because the candidate key does not exist, or because the candidate key is found but marked as a placeholder node, it is not yet known whether the virtual key exists or not. In this case, the system-scoped key name corresponding to the virtual key name is identified as the candidate key name (step 830). In other words, the candidate key name is exactly the virtual key name. If the candidate key does not exist (step 832), an error condition indicating the virtual key was not found is returned to the requestor (step 834). If on the other hand the candidate key exists (step 832), the request is checked to determine if the open request indicates an intention to modify the key (step 828). If not, the candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820). If, however, in step 828, it is determined that the open request indicates intention to modify the key, permission data associated with the key is checked to determine if modification of the key is allowed (step 836). If not, an error condition is returned to the requestor (step 838) indicating that modification of the key is not allowed. If the permission data indicates that the key may be modified, the

candidate key is copied to the user isolation scope (step 840). In some embodiments, the candidate key is copied to a location defined by the rules engine. For example, a rule may specify that the key is copied to an application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the key should be copied. Any ancestors of the requested key that do not appear in the isolation scope are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy. The newly copied scoped instance is identified as the literal key (step 842) and a request issued to open the literal key and the result returned to the requestor (step 820).

Still referring to FIG. 8 and now in more detail, a request to open a virtual registry key is received or intercepted (step 802). The corresponding literal registry key may be of user isolation scope, application isolation scope or system scope, or it may be scoped to an application isolation sub-scope or a user isolation sub-scope. In some embodiments, the request is hooked by a function that replaces the operating system function or functions for opening a registry key. In another embodiment a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for native registry keys. For embodiments in which a separate operating system function is provided for each type of registry key operation, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of registry key operations.

The request contains a registry key name, which is treated as a virtual registry key name by the isolation environment. The processing rule applicable

to the registry key open request is determined (step 804) by consulting the rules engine. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual registry key name provided for the requested registry key is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular registry key and, in these embodiments, the rule having the longest prefix match with the virtual registry key name is the rule applied to the request. In other embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 8 as a single database transaction or single lookup into a file, the rule lookup may be performed as a series of rule lookups .

If the rule action is "redirect" (step 806), the virtual registry key name provided in the request is mapped to the literal registry key name according to the applicable rule (step 808). A request to open the literal registry key using the literal registry key name is passed to the operating system and the result from the operating system is returned to the requestor (step 810). For example, a request to open a registry key named "registry_key_1" may result in the opening of a literal registry key named "Different_ registry_key _1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. In other embodiments, a registry filter driver facility conceptually similar to a file system filter driver facility may be provided by the operating system. In these embodiments, opening the literal registry key may be achieved by responding to the original request to open the virtual key by signaling to the registry filter manager to reparse the request using the determined literal key name.. If instead the rule action is "ignore" (step 806), then the literal registry key name is

determined to be exactly the virtual registry key name (step 812), and the request to open the literal registry key is passed to the operating system and the result from the operating system is returned to the requestor (step 810). For example, a request to open a registry key named "registry_key_1" will result in the opening of a literal registry key named "registry_key _1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. In another embodiment, this is accomplished by signaling to the registry filter manager to continue processing the original unmodified request in the normal fashion.

If in step 806 the rule action is "isolate", then the user-scoped registry key name corresponding to the virtual registry key name is identified as the candidate registry key name (step 814). In other words, the candidate registry key name is formed by mapping the virtual registry key name to the corresponding native registry key name specific to the applicable user isolation scope. For example, a request to open a registry key named "registry_key_1" may result in the opening of a literal registry key named "Isolated_UserScope_UserA_registry_key_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. In other embodiments, opening the literal registry key may be achieved by responding to the original request to open the virtual key by signaling to the registry filter manager to reparse the request using the determined literal key name.

In some embodiments, the literal name formed in order to isolate a requested virtual registry key may be based on the virtual registry key name received and a scope-specific identifier. The scope-specific identifier may be an identifier associated with an application isolation scope, a user isolation scope, a session isolation scope, an application isolation sub-scope, a user isolation sub-scope, or some combination of the above. The scope-specific identifier is used to "mangle" the virtual name received in the request.

In other embodiments, the user isolation scope or a sub-scope may be a registry key under which all keys that exist in the user isolation scope are stored. In some of these embodiments, the key hierarchy under the user isolation key reflects the path of the requested resource. In other words, the literal key path is formed by mapping the virtual key path to the user isolation scope. For example, if the requested key is HKLM\Software\Citrix\MyKey and the user isolation scope key is HKCU\Software\UserScope\, then the path to the user-scoped literal key may be HKCU\Software\UserScope\HKLM\Software\Citrix\MyKey. In other embodiments, the path to the user-scoped literal may be defined in a native naming convention. For example, the path to the user-scoped literal key may be HKCU\Software\UserScope\Registry\Machine\Software\Citrix\MyKey. In still other embodiments, the user-scoped keys may all be stored under a single key with names chosen to be unique and a database may be used to store the mapping between the requested key name and the name of the corresponding literal key stored in the user isolation key. In still other embodiments, the contents of the literal keys may be stored in a database or a file store.

The category of existence of the candidate key is determined by examining the user isolation scope and any metadata associated with the candidate key (step 816). If the candidate key is determined to have "negative existence", because either the candidate key or one of its ancestor keys in the user isolation scope is marked as deleted, this means the requested virtual key is known to not exist. In this case, an error condition indicating the requested key is not found is returned to the requestor (step 822).

In some embodiments, the literal registry key may be associated with metadata indicating that the virtualized registry key has already been deleted. In some embodiments, metadata about a registry key may be stored in a distinguished value held by that key, with the existence of that value hidden from ordinary application usage of registry APIs. In some embodiments, small amounts of metadata about a registry key may be stored directly in the literal key

name, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the key by virtual name check for possible variations of the literal key name due to the presence of a metadata indicator, and requests to retrieve the name of the key itself are hooked or intercepted in order to respond with the literal name. In other embodiments, the metadata indicator may be encoded in a subkey name or a registry value name instead of the key name itself. In still other embodiments, a registry key system may directly provide the ability to store some 3rd party metadata for each key. In some embodiments, metadata is stored in a database or other repository separate from the registry database. In some embodiments, a separate sub-scope may be used to store keys that are marked as deleted. The existence of a key in the sub-scope indicates that the key is marked as deleted.

In specific ones of these embodiments, a list of deleted keys or key system elements may be maintained and consulted to optimize this check for deleted keys. In these embodiments, if a deleted key is recreated then the key name may be removed from the list of deleted keys. In others of these embodiments, a key name may be removed from the list if the list grows beyond a certain size.

If instead in step 816 the candidate key is determined to have “positive existence”, because the candidate key exists in the user isolation scope and is not marked as a placeholder node, then the requested virtual key is known to exist. The candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820).

If, however, in step 816, the candidate key has “neutral existence” because the candidate key does not exist, or the candidate key exists but is marked as a placeholder node, it is not yet known whether the virtual key exists

or not. In this case the application-scoped key name corresponding to the virtual key name is identified as the candidate key name (step 824). In other words, the candidate key name is formed by mapping the virtual key name to the corresponding native key name specific to the applicable application isolation scope. The category of existence of the candidate key is determined by examining the application isolation scope and any metadata associated with the candidate key (step 826).

If the application-scoped candidate key is determined to have "negative existence", because either the candidate key or one of its ancestor keys in the application isolation scope is marked as deleted, this means the requested virtual key is known to not exist. In this case, an error condition indicating the requested key is not found is returned to the requestor (step 822).

If, however, in step 826 the candidate key is determined to have "positive existence", because the candidate key exists in the application isolation scope and is not marked as a placeholder node, then the requested virtual key is known to exist. The request is checked to determine if the open request indicates an intention to modify the key (step 828). If not, the candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820).

If, however, in step 828, it is determined that the open request indicates intention to modify the key, permission data associated with the key is checked to determine if modification of the key is allowed (step 836). In some embodiments, the permission data is associated with the application-scoped candidate key. In some of these embodiments, the permissions data is stored in a rules engine or in metadata associated with the candidate key. In other embodiments, the permission data associated with the candidate key is provided by the operating system. Further, the rules engine may include configuration settings instructing the isolation environment to obey or override the native permission data for virtualized copies of resources. In some embodiments, the rules may specify for

some virtual resources the scope in which modifications are to occur, for example the system scope or the application isolation scope or a sub-scope, or the user isolation scope or a sub-scope. In some embodiments, the rules engine may specify configuration settings that apply to subsets of the virtualized native resources based on hierarchy. In some of these embodiments, the configuration settings may be specific to each atomic native resource.

If the permission data associated with the candidate key indicates that it may not be modified, an error condition is returned to the requestor (step 838) indicating that modification of the key is not allowed. If the permission data indicates that the key may be modified, the candidate key is copied to the user isolation scope (step 840). In some embodiments, the candidate key is copied to a location defined by the rules engine. For example, a rule may specify that the key is copied to another application isolation scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the key should be copied. Any ancestors of the requested key that do not appear in the isolation scope to which the key is copied are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy.

In some embodiments, metadata is associated with keys copied to the isolation scope that identifies the date and time at which the keys were copied. This information may be used to compare the time stamp associated with the copied instance of the key to the time stamp of the last modification of the original instance of the key or of another instance of the key located in a lower isolation scope. In these embodiments, if the original instance of the key, or an instance of the key located in a lower isolation scope, is associated with a time stamp that is later than the time stamp of the copied key, that key may be copied to the isolation scope to update the candidate key. In other embodiments, the copy of the key in the isolation scope may be associated with metadata identifying the scope containing the original key that was copied.

In further embodiments, keys that are copied to isolation scopes because they have been opened with intent to modify them may be monitored to determine if they are, in fact, modified. In one embodiment a copied key may be associated with a flag that is set when the key is actually modified. In these embodiments, if a copied key is not actually modified, it may be removed from the scope to which it was copied after it is closed, as well as any placeholder nodes associated with the copied key.

The scoped instance is identified as the literal key (step 842) and a request issued to open the literal key and the result returned to the requestor (step 820).

Returning to step 826, if the candidate key has neutral existence because the candidate key does not exist, or if the candidate key is found but marked as a placeholder node, it is not yet known whether the virtual key exists or not. In this case, the system-scoped key name corresponding to the virtual key name is identified as the candidate key name (step 830). In other words, the candidate key name is exactly the virtual key name.

If the candidate key does not exist (step 832), an error condition indicating the virtual key was not found is returned to the requestor (step 834). If on the other hand the candidate key exists (step 832), the request is checked to determine if the open request indicates an intention to modify the key (step 828).

As above, if the candidate key is being opened without the intent to modify it, the system-scoped candidate key is identified as the literal key for the request (step 818), and a request issued to open the literal key and the result returned to the requestor (step 820). If, however, in step 828, it is determined that the open request indicates intention to modify the key, permission data associated with the key is checked to determine if modification of the key is allowed (step 836). In some embodiments, the permission data is associated with the application-scoped candidate key. In some of these embodiments, the permissions data is stored in a rules engine or in metadata associated with the candidate key. In

other embodiments, the permission data associated with the candidate key is provided by the operating system. Further, the rules engine may include configuration settings instructing the isolation environment to obey or override the native permission data for virtualized copies of resources. In some embodiments, the rules may specify for some virtual resources the scope in which modifications are to occur, for example the system scope or the application isolation scope or a sub-scope, or the user isolation scope or a sub-scope. In some embodiments, the rules engine may specify configuration settings that apply to subsets of the virtualized native resources based on hierarchy. In some of these embodiments, the configuration settings may be specific to each atomic native resource.

If the permission data associated with the system-scoped candidate key indicates that the key may not be modified, an error condition is returned to the requestor (step 838) indicating that modification of the key is not allowed. If, however, the permission data indicates that the key may be modified, the candidate key is copied to the user isolation scope (step 840). In some embodiments, the candidate key is copied to a location defined by the rules engine. For example, a rule may specify that the key is copied to an application isolation scope or that it may be left in the system scope. In other embodiments the rules may specify a particular application isolation sub-scope or user isolation sub-scope to which the key should be copied. Any ancestors of the requested key that do not appear in the isolation scope are created as placeholders in the isolation scope in order to correctly locate the copied instance in the hierarchy.

In some embodiments, metadata is associated with keys copied to the isolation scope that identifies the date and time at which the keys were copied. This information may be used to compare the time stamp associated with the copied instance of the key to the time stamp of the last modification of the original instance of the key. In these embodiments, if the original instance of the key is associated with a time stamp that is later than the time stamp of the copied

key, the original key may be copied to the isolation scope to update the candidate key. In other embodiments, the candidate key copied to the isolation scope may be associated with metadata identifying the scope from which the original key was copied.

In further embodiments, keys that are copied to isolation scopes because they have been opened with intent to modify them may be monitored to determine if they are, in fact, modified. In one embodiment a copied key may be associated with a flag that is set when the key is actually modified. In these embodiments, if a copied key is not actually modified, when it is closed it may be removed from the scope to which it was copied, as well as any placeholder nodes associated with the copied key. In still further embodiments, the key is only copied to the appropriate isolation scope when the key is actually modified.

The scoped instance is identified as the literal key (step 842) and a request issued to open the literal key and the result returned to the requestor (step 820).

4.2.2 Registry Key Delete Operations

Referring now to FIG. 9, and in brief overview, one embodiment of the steps taken to delete a registry key is depicted. Before a key can be deleted, the key must first be opened successfully with delete access (step 901). If the key is not opened successfully, an error is returned (step 916). If the virtual key is opened successfully, a request to delete a virtualized registry key is received or intercepted, the request including the handle to the literal key corresponding to the virtual key (step 902). A rule determines how the registry key operation is processed (step 904). In addition to the rule applicable to the key to be deleted, any other rules applicable to immediate subkeys are examined (step 905). For each rule applicable to an immediate subkey found, an attempt is made to open a virtual subkey, with the virtual subkey's name being specified by the name given in the rule found in step 905. If a subkey with a name corresponding to one of the rules found in step 905 is opened successfully (step 906), then the virtual

key is considered to have subkeys, which means it cannot be deleted, and an error returned (step 907).

If, after all the virtual key names extracted in step 905 have been attempted to be opened (step 906), no virtual keys were found to exist, further examination is required. If the rule action is not “isolate”, but is “redirect”, or is “ignore” (step 908),, a request to delete the literal registry key is passed to the operating system and the result from the operating system is returned to the requestor (step 911). If however the rule action determined in step 908 is “isolate” the aggregated virtualized registry key is consulted to determine if it contains any virtual subkeys (step 914). If the virtualized key has virtual subkeys, then the deletion cannot continue, and an error is returned indicating the key has not been deleted (step 920). If the virtualized key does not have virtual subkeys, then the literal key corresponding to the virtual key is examined to determine if it masks a scoped key with the same virtual name in another scope level (step 922). If the literal key corresponding to the virtual key does not mask a differently scoped key with the same virtual name, then the literal key which corresponds to the virtual key is deleted, and the result returned (step 926). If the literal key corresponding to the virtual key masks a differently scoped key with the same virtual name, then the literal key corresponding to the virtual key is marked with a value indicating that it is deleted, and a successful result returned to the caller (step 924).

Still referring to FIG. 9, and in more detail, in order to delete a key, it must first be opened with delete access (step 901). The request to open the key with delete access includes the name of the key which is treated as a virtual name by the isolation environment. A full virtualized key open is performed as described in section 4.2.1. If the virtualized open operation fails, an error is returned to the requestor (step 916). If the virtualized open operation succeeds, the handle of the literal key corresponding to the virtual key is returned to the requestor. Subsequently a request to delete the registry key which was opened in step 901

is received or intercepted (step 902). The opened literal registry key may be of user isolation scope, application isolation scope, system scope, or some applicable isolation sub-scope. In some embodiments, the delete request is hooked by a function that replaces the operating system function or functions for deleting the registry key. In another embodiment a hooking dynamically-linked library is used to intercept the delete request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for native registry keys. In other embodiments, a registry filter driver facility conceptually similar to a file system filter driver facility may be provided by the operating system. A practitioner skilled in the art may create a registry filter driver to which the operating system passes requests to perform registry operations, thus providing a mechanism to intercept registry operation requests. For embodiments in which a separate operating system function is provided for each type of registry key function, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of registry key functions.

The delete request contains a literal key handle. The virtual key name associated with the handle is determined by querying the operating system for the literal name associated with the handle. The rules engine is consulted to determine the virtual name associated with the literal name, if any. A rule determining how the registry key operation is processed (step 904) is obtained by consulting the rules engine. In some embodiments, the virtual key name of the virtual registry key to be deleted is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular virtual registry key and, in some of

these embodiments, the rule having the longest prefix match with the virtual key name is the rule applied to the request. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat registry key database. In some embodiments, the virtual key name corresponding to the virtual key handle in the request is used as an index into a rules engine to locate one or more rules that apply to the request. In some embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. The rule lookup may occur as a series of decisions, or the rule lookup may occur as a single database transaction.

The virtual name of the key to be deleted is used to consult the rules engine to locate the set of rules applicable to any immediate child keys of the virtual key to delete, but not applicable to the virtual key to be deleted. This set of rules is located whether those child keys exist or not (step 905). If this set of rules applicable to immediate child keys is not empty, then the virtual name of each of these rules is extracted. An attempt is made to do a full virtualized open of each of the virtual child key names extracted, in turn (step 906). If any of the virtual keys corresponding to any of these virtual names can be opened successfully, then this means that a virtual subkey exists. This means that the virtual key cannot be deleted, as it has a virtual child that exists, and an error is returned (step 907). If after examining all of the set of rules applicable to immediate children of the virtual key (step 905), no virtual subkeys are found to exist, the deletion can continue. For example, a key with virtual name “key_1” may have child rules applicable to “key1\subkey_1” and “key1\subkey_2”. In this step, an attempt is made to do a virtualized open of “key1\subkey_1” and “key1\subkey_2”. If either of these virtual subkeys can be opened successfully,

then the deletion will fail, and an error is returned (step 907). Only if neither of these virtual subkeys exist can the deletion continue.

If the rule action is not "isolate", but is "redirect", or is "ignore" (step 908), a request to delete the literal registry key using the literal key handle is passed to the operating system and the result from the operating system is returned to the requestor (step 911). This request will fail if the literal key contains literal subkeys. In one embodiment, the request to delete the literal registry key is accomplished by calling the original version of the hooked function and passing the literal key handle to the function as an argument. In embodiments that make use of a registry filter driver, this is accomplished by responding to the request with a completion status that signals the operating system to perform normal processing on the request. In some embodiments, operating system permissions associated with the literal registry key may prevent its deletion. In these embodiments, an error message is returned that the virtual registry key could not be deleted.

If the rule action determined in step 908 is "isolate", then the aggregated virtualized registry key is consulted to determine if it contains any virtual subkeys (step 914). If the requested virtual registry key contains virtual subkeys, then the virtual key cannot be deleted, and an error is returned to the caller (step 920).

If the requested virtual registry key does not contain virtual subkeys, then the virtual key can be deleted. The action taken next depends on the scope that contains the literal key to be deleted. For example, a request to delete a virtual registry key may result in the deletion of an application-scoped literal key. The scope containing the literal key can be determined by consulting the rules engine with the full path to the literal key.

If the literal key to be deleted is found in a particular scope, and that literal key masks another key of the same virtual name in another scope, then the literal key to be deleted is marked as deleted, and a result returned to the requestor (step 924). For example, a virtual key that corresponds to a user-scoped literal

key is considered to mask a differently-scoped key if a corresponding application-scoped key with the same virtual name or a corresponding system-scoped key with the same virtual name has "positive existence", that is, exists in the scope, and is not marked as a placeholder, and is not considered to be deleted. Similarly, an application-scoped key is considered to mask a system-scoped key corresponding to the same virtual name if that system-scoped key exists and is not considered to be deleted.

If the literal key to be deleted is found not to mask another key of the same virtual name in another scope, then the literal key to be deleted is actually deleted and a result returned (step 926).

In some embodiments, operating system permissions associated with the literal registry key may prevent deletion of the literal registry key. In these embodiments, an error message is returned that the virtual registry key could not be deleted.

In some embodiments, the literal registry key may be associated with metadata indicating that the virtualized registry key has already been deleted. In some embodiments, metadata about a registry key may be stored in a distinguished value held by that key, with the existence of that value hidden from ordinary application usage of registry APIs. In some embodiments, small amounts of metadata about a registry key may be stored directly in the literal key name, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the key by virtual name check for possible variations of the literal key name due to the presence of a metadata indicator, and requests to retrieve the name of the key itself are hooked or intercepted in order to respond with the literal name. In other embodiments, the metadata indicator may be encoded in a subkey name or a registry value name instead of the key name itself. In still other embodiments, a registry key system may

directly provide the ability to store some 3rd party metadata for each key. In some embodiments, metadata could be stored in a database or other repository separate from the registry database. In some embodiments, a separate sub-scope may be used to store keys that are marked as deleted. The existence of a key in the sub-scope indicates that the key is marked as deleted.

In specific ones of these embodiments, a list of deleted keys or key system elements may be maintained and consulted to optimize this check for deleted keys. In these embodiments, if a deleted key is recreated then the key name may be removed from the list of deleted keys. In others of these embodiments, a key name may be removed from the list if the list grows beyond a certain size.

In some embodiments, an ancestor of the literal registry key in the same scope is associated with metadata indicating that it is deleted, or is otherwise indicated to be deleted. In these embodiments, an error message may be returned indicating that the virtualized registry key does not exist. In specific ones of these embodiments, a list of deleted registry keys or registry key system elements may be maintained and consulted to optimize this check for deleted registry keys.

4.2.3 Registry Key Enumeration Operations

Referring now to FIG. 10, and in brief overview, one embodiment of the steps taken to enumerate a key in the described virtualized environment is shown. Before a key can be enumerated, the key must first be opened successfully with enumerate access (step 1001). If the key is not opened successfully, an error is returned (step 1040). If the virtual key is opened successfully, a request to enumerate is received or intercepted, the request including the handle to the literal key corresponding to the virtual key (step 1002). The virtual key name corresponding to the handle is determined, and the rules engine is consulted to determine the rule for the key specified in the enumerate request (step 1004). If the rule doesn't specify an action of "isolate", but instead

specifies "ignore" or specifies "redirect" (step 1006), the literal key identified by the literal key handle is enumerated, and the enumeration results stored in a working data store (step 1012), followed by step 1030 as described later.

If, however, the rule action specifies "isolate," firstly the system scope is enumerated; that is, the candidate key name is exactly the virtual key name, and if the candidate key exists it is enumerated. The enumeration results are stored in a working data store. If the candidate key does not exist, the working data store remains empty at this stage (step 1014). Next, the candidate key is identified as the application-scoped instance of the virtual key, and the category of existence of the candidate key is determined (step 1015). If the candidate key has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 1042). If instead the candidate key does not have negative existence, the candidate key is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each subkey in the enumeration, its category of existence is determined. Subkeys with negative existence are removed from the working data store, and subkeys with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding subkey if one is already present in the working data store (step 1016).

In either case, the candidate key is identified as the user-scoped instance of the virtual key, and the category of existence of the candidate key is determined (step 1017). If the candidate key has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 1044). If instead the candidate key does not have negative existence, the candidate key is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each subkey in the enumeration, its

category of existence is determined. Subkeys with negative existence are removed from the working data store, and subkeys with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding subkey if one is already present in the working data store (step 1018), followed by step 1030 as described below.

Then, for all three types of rules, step 1030 is executed. The rules engine is queried to find the set of rules whose filters match immediate children of the requested virtual key name, but do not match the requested virtual key name itself (step 1030). For each rule in the set, the existence of the virtual child whose name matches the name in the rule is determined. If the child has positive existence, it is added to the working data store, replacing any child of the same name already there. If the child has negative existence, the entry in the working data store corresponding to the child, if any, is removed. (Step 1032). Finally, the constructed enumeration is then returned from the working data store to the requestor (step 1020).

Still referring to FIG. 10, and in more detail, in order to enumerate a key, it must first be opened with enumerate access (step 1001). The request to open the key with enumerate access includes the name of the key which is treated as a virtual name by the isolation environment. A full virtualized key open is performed as described in section 4.2.1. If the virtualized open operation fails, an error is returned to the requestor (step 1040). If the virtualized open operation succeeds, the handle of the literal key corresponding to the virtual key is returned to the requestor. Subsequently a request to enumerate the registry key which was opened in step 1001 is received or intercepted (step 1002). The opened literal registry key may be of user isolation scope, application isolation scope, system scope, or some applicable isolation sub-scope. In some embodiments, the enumerate request is hooked by a function that replaces the operating system function or functions for enumerating a registry key. In another

embodiment a hooking dynamically-linked library is used to intercept the enumerate request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for native registry keys. In other embodiments, a registry filter driver facility conceptually similar to a file system filter driver facility may be provided by the operating system. A practitioner skilled in the art may create a registry filter driver to which the operating system passes requests to perform registry operations, thus providing a mechanism to intercept registry operation requests. For embodiments in which a separate operating system function is provided for each type of registry key function, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of registry key functions.

The enumerate request contains a literal key handle. The virtual key name associated with the handle is determined by querying the operating system for the literal name associated with the handle. The rules engine is consulted to determine the virtual name associated with the literal name, if any.

A rule determining how the registry key operation is processed (step 1004) is obtained by consulting the rules engine. In some embodiments, the virtual key name of the virtual registry key to be enumerated is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular virtual registry key and, in some of these embodiments, the rule having the longest prefix match with the virtual key name is the rule applied to the request. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash

table, or a flat registry key database. In some embodiments, the virtual key name corresponding to the virtual key handle in the request is used as an index into a rules engine to locate one or more rules that apply to the request. In some embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. The rule lookup may occur as a series of decisions, or the rule lookup may occur as a single database transaction.

If the rule action is not "isolate" (step 1006), but is "ignore" or is "redirect", then a request to enumerate the literal key is passed to the operating system using the literal key handle, and the enumeration results, if any, are stored in the working data store (step 1012), and step 1030 is executed as described later. In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. In other embodiments, a registry filter driver facility conceptually similar to a file system filter driver facility may be provided by the operating system. In these embodiments, enumerating the literal registry key may be achieved by responding to the original request to enumerate the key by signaling to the registry filter manager to process the unmodified request in the normal fashion.

If the rule action determined in step 1010 is "isolate", then the system scope is enumerated. To achieve this, the candidate key is identified as the system-scoped key corresponding to the virtual key to be enumerated. The candidate key is enumerated, and the results of the enumeration are stored in a working data store (step 1014). In some embodiments, the working data store is comprised of a memory element. In other embodiments, the working data store comprises a database or a key or a solid-state memory element or a persistent data store.

Next, the candidate key is identified as the application-scoped instance of the virtual key, and the category of existence of the candidate key is determined (step 1015). If the candidate key has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 1042).

In some embodiments, the candidate registry key may be associated with metadata indicating that the candidate registry key has been deleted. In some embodiments, metadata about a registry key may be stored in a distinguished value held by that key, with the existence of that value hidden from ordinary application usage of registry APIs. In some embodiments, small amounts of metadata about a registry key may be stored directly in the literal key name, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the key by virtual name check for possible variations of the literal key name due to the presence of a metadata indicator, and requests to retrieve the name of the key itself are hooked or intercepted in order to respond with the literal name. In other embodiments, the metadata indicator may be encoded in a subkey name or a registry value name instead of the key name itself. In still other embodiments, a registry key system may directly provide the ability to store some 3rd party metadata for each key. In some embodiments, metadata is stored in a database or other repository separate from the registry database. In some embodiments, a separate sub-scope may be used to store keys that are marked as deleted. The existence of a key in the sub-scope indicates that the key is marked as deleted.

If instead, in step 1015, the candidate key does not have negative existence, the candidate key is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each subkey in the enumeration, its category of existence is determined. Subkeys with negative

existence are removed from the working data store, and subkeys with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding subkey if one is already present in the working data store (step 1016).

In either case, the candidate key is identified as the user-scoped instance of the virtual key, and the category of existence of the candidate key is determined (step 1017). If the candidate key has "negative existence", i.e. it or one of its ancestors in the scope is marked as deleted, then within this scope it is known to be deleted, and this is indicated by flushing the working data store (step 1044). If instead the candidate key does not have negative existence, the candidate key is enumerated and any enumeration results obtained are merged into the working data store. In particular, for each subkey in the enumeration, its category of existence is determined. Subkeys with negative existence are removed from the working data store, and subkeys with positive existence, i.e. those that exist and are not marked as placeholders and are not marked as deleted, are added to the working data store, replacing the corresponding subkey if one is already present in the working data store (step 1018), followed by step 1030 as described below.

Then, for all three types of rules, step 1030 is executed. The rules engine is queried to find the set of rules whose filters match immediate children of the requested key, but do not match the requested key itself (step 1030). For each rule in the set, the existence of the virtual child whose name matches the name in the rule is determined. In some embodiments, this is determined by examining the appropriate isolation scope and the metadata associated with the virtual child. In other embodiments, this is determined by attempting to open the key. If the open request succeeds, the virtual child has positive existence. If the open request fails with an indication that the virtual child does not exist, the virtual child has negative existence.

If the child has positive existence, it is added to the working data store, replacing any child of the same name already there. If the child has negative existence, the child in the working data store corresponding to the virtual child, if any, is removed. (Step 1032). Finally, the constructed enumeration is then returned from the working data store to the requestor (step 1020).

A practitioner of ordinary skill in the art will realize that the layered enumeration process described above can be applied with minor modification to the operation of enumerating a single isolation scope which comprises a plurality of isolation sub-scopes. A working data store is created, successive sub-scopes are enumerated and the results are merged into the working data store to form the aggregated enumeration of the isolation scope.

4.2.4. Registry Creation Operations

Referring now to FIG. 11, and in brief overview, one embodiment of the steps taken to create a key in the isolation environment is shown. A request to create a key is received or intercepted (step 1102). The request contains a key name, which is treated as a virtual key name by the isolation environment. An attempt is made to open the requested key using full virtualization using applicable rules, i.e. using appropriate user and application isolation scope, as described in section 4.2.1 (step 1104). If access is denied (step 1106), an access denied error is returned to the requestor (step 1109). If access is granted (step 1106), and the requested key is successfully opened (step 1110), the requested key is returned to the requestor (step 1112). However, if access is granted (step 1106), but the requested key is not opened successfully (step 1110) then if the parent of the requested key also does not exist (step 1114), an error appropriate to the request semantics is issued to the requestor (step 1116). If on the other hand, the parent of the requested key is found in full virtualized view using the appropriate user and application scope (step 1114), a rule then determines how the key operation is processed (step 1118). If the rule action is “redirect” or “ignore” (step 1120), the virtual key name is mapped directly to a literal key name

according to the rule. Specifically, if the rule action is “ignore”, the literal key name is identified as exactly the virtual key name. If, instead, the rule action is “redirect”, the literal key name is determined from the virtual key name as specified by the rule. Then a request to create the literal key is passed to the operating system, and the result is returned to the requestor (step 1124). If on the other hand, the rule action determined in step 1120 is “isolate”, then the literal key name is identified as the instance of the virtual key name in the user isolation scope. If the literal key already exists, but is associated with metadata indicating that it is a placeholder or that it is deleted, then the associated metadata is modified to remove those indications, and it is ensured that the key is empty. In either case, a request to open the literal key is passed to the operating system (step 1 126). If the literal key was opened successfully (step 1128), the literal key is returned to the requestor (step 1130). If on the other hand, in step 1128, the requested key fails to open, placeholders for each ancestor of the literal key that does not currently exist in the user-isolation scope (step 1132) and a request to create the literal key using the literal name is passed to the operating system and the result is returned to the requestor (step 1134).

Still referring to FIG. 11, and in more detail, a request to create a key is received or intercepted (step 1102). In some embodiments, the request is hooked by a function that replaces the operating system function or functions for creating the key. In another embodiment, a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for key operations. For embodiments in which a separate operating system function is provided for each type of key operation, each function may be hooked

separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of key operations.

The request contains a key name, which is treated as a virtual key name by the isolation environment. In some embodiments, the virtual key name may be expressed as a combination of a handle to a parent key, and the relative path name to the descendant key. The parent key handle is associated with a literal key name, which is itself associated with a virtual key name. The requestor attempts to open the virtual key using full virtualization using applicable rules, i.e. using appropriate user and application isolation scope, as described in section 4.2.1 (step 1104). If access is denied during the full virtualized open operation (step 1106), an access denied error is returned to the requestor (step 1109). If access is granted (step 1106), and the requested virtual key is successfully opened (step 1110), the corresponding literal key is returned to the requestor (step 1112). However, if access is granted (step 1106), but the virtual key is not opened successfully (step 1110) then the virtual key has been determined not to exist. If the virtual parent of the requested virtual key also does not exist, as determined by the procedures in section 4.2.1 (step 1114), an error appropriate to the request semantics is issued to the requestor (step 1116). If on the other hand, the virtual parent of the requested virtual key is found in full virtualized view using the appropriate user and application scope (step 1114), then a rule that determines how the create operation is processed is located (step 1118) by consulting the rules engine. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat key database. In some embodiments, the virtual key name provided for the requested key is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular key and, in some of these embodiments, the rule having the longest prefix match with the virtual key name is the rule applied to the request. In some

embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 11 as a single database transaction or single lookup into a key, the rule lookup may be performed as a series of rule lookups..

If the rule action is “redirect” or “ignore” (step 1120), the virtual key name is mapped directly to a literal key name according to the rule (step 1124). If the rule action is “redirect” (step 1120), the literal key name is determined from the virtual key name as specified by the rule (step 1124). If the rule action is “ignore” (step 1120), the literal key name is determined to be exactly the virtual key name (step 1124). If the rule action is “ignore” or the rule action is “redirect”, a request to create the literal key using the determined literal key name is passed to the operating system and the result from the operating system is returned to the requestor (step 1124). For example, a request to create a virtual key named “key_1” may result in the creation of a literal key named "Different_ key_1.” In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument. (step 1124). In other embodiments, a registry filter driver facility conceptually similar to a file system filter driver facility may be provided by the operating system. In these embodiments, creating the literal registry key may be achieved by responding to the original request to create the virtual key by signaling to the registry filter manager to reparse the request using the determined literal key name.

If the rule action determined in step 1120 is not “ignore” or “redirect” but is “isolate,” then the literal key name is identified as the instance of the virtual key name in the user isolation scope. If the literal key already exists, but is associated with metadata indicating that it is a placeholder or that it is deleted, then the associated metadata is modified to remove those indications, and it is ensured that the key is empty.

In some embodiments, metadata about a registry key may be stored in a distinguished value held by that key, with the existence of that value hidden from ordinary application usage of registry APIs. In some embodiments, small amounts of metadata about a registry key may be stored directly in the literal key name, such as by suffixing the virtual name with a metadata indicator, where a metadata indicator is a string uniquely associated with a particular metadata state. The metadata indicator may indicate or encode one or several bits of metadata. Requests to access the key by virtual name check for possible variations of the literal key name due to the presence of a metadata indicator, and requests to retrieve the name of the key itself are hooked or intercepted in order to respond with the literal name. In other embodiments, the metadata indicator may be encoded in a subkey name or a registry value name instead of the key name itself. In still other embodiments, a registry key system may directly provide the ability to store some 3rd party metadata for each key. In some embodiments, metadata could be stored in a database or other repository separate from the registry database. In some embodiments, a separate sub-scope may be used to store keys that are marked as deleted. The existence of a key in the sub-scope indicates that the key is marked as deleted.

In specific ones of these embodiments, a list of deleted keys or key system elements may be maintained and consulted to optimize this check for deleted keys. In these embodiments, if a deleted key is recreated then the key name may be removed from the list of deleted keys. In others of these embodiments, a key name may be removed from the list if the list grows beyond a certain size.

In either case, a request to open the user-scoped literal key is passed to the operating system (step 1126). In some embodiments, rules may specify that the literal key corresponding to the virtual key should be created in a scope other than the user isolation scope, such as the application isolation scope, the system scope, a user isolation sub-scope or an application isolation sub-scope.

If the literal key was opened successfully (step 1128), the literal key is returned to the requestor (step 1130). If on the other hand, in step 1128, the requested key fails to open, placeholders are created for each ancestor of the literal key that does not currently exist in the user-isolation scope (step 1132) and a request to create the literal key using the literal name is passed to the operating system and the result is returned to the requestor (step 1134).

This embodiment is for operating systems with APIs or facilities that only support creation of one level per call/invocation. Extension to multi-levels per call/invocation should be obvious to one skilled in the art.

4.3 Named Object Virtualization

Another class of system-scoped resources that may be virtualized using the techniques described above are named objects, which include semaphores, mutexes, mutants, waitable timers, events, job objects, sections, named pipes, and mailslots. These objects are characterized in that they typically exist only for the duration of the process which creates them. The name space for these objects may be valid over an entire computer (global in scope) or only in an individual user session (session scoped).

Referring now to FIG. 12, and in brief overview, a request to create or open a named object is received or intercepted (step 1202). That request contains an object name which is treated as a virtual name by the isolation environment. A rule determining how to treat the request is determined (step 1204). If the rule indicates that the request should be ignored (step 1206), the literal object name is determined to be the virtual name (step 1207), and a request to create or open the literal object is issued to the operating system (step 1214). If the determined rule is not to ignore the request, but indicates instead that the request should be redirected (step 1208), the literal object name is determined from the virtual name as specified by the redirection rule (step 1210) and a create or open request for the literal object is issued to the operating system (step 1214). If the rule does not indicate that the request should be

redirected (step 1208), but instead indicates that the request should be isolated, then the literal object name is determined from the virtual name as specified by the isolation rule (step 1212) and a create or open command for the literal object is issued to the operating system (step 1214). The handle of the literal object returned by the operating system in response to the issued create or open command is returned to the program requesting creation or opening of the virtual object (step 1216).

Still referring to FIG. 12, and in more detail, a request from a process to create or open a named object is intercepted (step 1202). The named object may be of session scope or it may be of global scope. In some embodiments, the request is hooked by a function that replaces the operating system function or functions for creating or opening the named object. In another embodiment a hooking dynamically-linked library is used to intercept the request. The hooking function may execute in user mode or in kernel mode. For embodiments in which the hooking function executes in user mode, the hooking function may be loaded into the address space of a process when that process is created. For embodiments in which the hooking function executes in kernel mode, the hooking function may be associated with an operating system resource that is used in dispatching requests for system objects. The request to create or open the named object may refer to any one of a wide variety of system-scoped resources that are used for interprocess communication and synchronization and that are identified by a unique identifier including semaphores, mutexes, mutants, waitable timers, file-mapping objects, events, job objects, sections, named pipes, and mailslots. For embodiments in which a separate operating system function is provided for each type of object, each function may be hooked separately. Alternatively, a single hooking function may be provided which intercepts create or open calls for several types of objects.

The intercepted request contains an object name which is treated as a virtual name by the isolation environment. A rule determining how to treat the

request for the object is determined (step 1204) by consulting the rules engine. In some embodiments, the rules engine may be provided as a relational database. In other embodiments, the rules engine may be a tree-structured database, a hash table, or a flat file database. In some embodiments, the virtual name provided for the requested object is used to locate in the rule engine a rule that applies to the request. In particular ones of these embodiments, multiple rules may exist in the rules engine for a particular object and, in these embodiments, the rule having the longest prefix match with the virtual name is the rule applied to the request. In some embodiments, a process identifier is used to locate in the rule engine a rule that applies to the request, if one exists. The rule associated with a request may be to ignore the request, redirect the request, or isolate the request. Although shown in FIG. 12 as a series of decisions, the rule lookup may occur as a single database transaction.

If the rule indicates that the request should be ignored (step 1206), the literal object name is determined to be the virtual name, and a request to create or open the literal object is issued to the operating system (step 1214). For example, a request to create or open a named object named "Object_1" will result in the creation of an actual object named "Object_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument.

If the rule determined by accessing the rules engine is not to ignore the request, but indicates instead that the request should be redirected (step 1208), the literal object name is determined from the virtual name as specified by the redirection rule (step 1210) and a create or open request for the literal object is issued to the operating system (step 1214). For example, a request to create or open a named object named "Object_1" may result in the creation of an actual object named "Different_Object_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument.

If the rule does not indicate that the request should be redirected (step 1208), but instead indicates that the request should be isolated, then the literal object name is determined from the virtual name as specified by the isolation rule (step 1212) and a create or open command for the literal object is issued to the operating system (step 1214). For example, a request to create or open a named object named "Object_1" may result in the creation of an actual object named "Isolated_Object_1". In one embodiment, this is accomplished by calling the original version of the hooked function and passing the formed literal name to the function as an argument.

The literal name formed in order to isolate a requested system object may be based on the virtual name received and a scope-specific identifier. The scope-specific identifier may be an identifier associated with an application isolation scope, a user isolation scope, a session isolation scope, or some combination of the three. The scope-specific identifier is used to "mangle" the virtual name received in the request. For example, if the request for the named object "Object_1" is isolated for the application-isolation scope whose associated identifier is "SA1", the literal name may be "Isolated_AppScope_SA1_Object_1". The following table identifies the effect of mangling the name of an object with session isolation scope, or user isolation scope, and an application isolation scope. Mangling with combinations of scopes combines the restrictions listed in the table.

	Session-specific identifier	User-specific identifier	Application-specific identifier
Global object	Object available to all isolated applications executing in the context of the user session	Object available to all isolated applications executing on behalf of the user	Object available to all isolated applications executing in application isolation scope

Session object	Object available to all isolated applications executing in the context of the user session	Object available to all isolated applications executing in the session on behalf of the user	Object available to all isolated applications executing in the application isolation scope within the session

For embodiments in which the operating system is one of the WINDOWS family of operating systems, object scope may be modified by toggling the global/local name prefix associated with the object, which, for isolated applications, has the same effect as mangling the object name with a session-specific identifier. However, toggling the global/local name prefix also affects the object scope for non-isolated applications.

The handle of the literal object returned by the operating system in response to the command issued in step 1214 to create or open the named object is returned to the program requesting creation or opening of the virtual object (step 1216).

4.4 Window Name Virtualization

Other classes of system-scoped resources that may be virtualized using the techniques described above are window names and window class names. Graphical software applications use the name of a window or its window class as a way of identifying if an application program is already running and for other forms of synchronization. Referring now to FIG. 13, and in brief overview, a request concerning a window name or window class is received or intercepted (step 1302). A request can be in the form of a Win32 API call or in the form of a Window message. Both types of requests are handled. Those requests contain, or request retrieval of, window names and/or window class names that are treated as virtual names by the isolation environment. If the request is to retrieve

a window name or window class for a window identified by a handle (step 1304), a window mapping table is consulted to determine if the handle and the requested information concerning the window is known (step 1306). If so, the requested information from the window mapping table is returned to the requestor (step 1308). If not, the request is passed to the operating system (step 1310), and the result returned to the requestor (step 1314). If, in step 1304, the request provides a window name or window class, the request is checked to determine if it specifies one of a class of windows defined by the operating system (step 1320). If it does, the request is issued to the operating system and the result returned from the operating system is returned to the requestor (step 1322). If the request does not specify one of a class of windows defined by the operating system, the literal class name is determined based on the virtual class name and the rules (step 1324) and the literal window name is determined based on the virtual window name and the rules (step 1326). The request is then passed to the operating system using the literal window and literal class names (step 1328). If either the literal window name or literal window class name determined in steps 1324 and 1326 differ from the corresponding virtual name, then the window mapping table entry for the window handle is updated to record the virtual window name or virtual class name provided in the request (step 1330). If the response from the operating system includes native window names or native identifications of classes, those are replaced with the virtual window name or virtual class name provided in the request (step 1312) and the result is returned to the requestor (step 1314).

Still referring to FIG. 13, and in more detail a request concerning a window name or window class is received or intercepted (step 1302). Those requests contain or request retrieval of window names and/or window class names that are treated as virtual names by the isolation environment.

If the request is to retrieve a window name or window class for a window identified by a handle (step 1304), a window mapping table is consulted to

determine if the handle and the requested information concerning the window is known (step 1306). In some embodiments, instead of a mapping table, additional data is stored for each window and window class using facilities provided by the operating system.

If so, the requested information from the window mapping table is returned to the requestor (step 1308). If not, the request is passed to the operating system (step 1310), and the result returned to the requestor (step 1314).

If, in step 1304, the request provides a window name or window class, the request is checked to determine if it specifies one of a class of windows defined by the operating system (step 1320). If it does, the request is passed to the operating system and the result returned from the operating system is returned to the requestor (step 1322).

If the request does not specify one of a class of windows defined by the operating system, the literal class name is determined based on the virtual class name and the rules (step 1324) and the literal window name is determined based on the virtual window name and the rules (step 1326). The request is then passed to the operating system using the literal window and literal class names (step 1328). In some embodiments the window names and window class names may be atoms, rather than character string literals. Typically an application places a string in an atom table and receives a 16-bit integer, called an atom, that can be used to access the string.

If either the literal window name or literal window class name determined in steps 1324 and 1326 differ from the corresponding virtual name, then the window mapping table entry for the window handle is updated to record the virtual window name or virtual class name provided in the request (step 1330).

If the response from the operating system includes native window names or native identifications of classes, those are replaced with the virtual window name or virtual class name provided in the request (step 1312) and the result is returned to the requestor (step 1314).

Referring now to FIG 13A, the literal window name or window class name is determined as shown there. The rules engine is consulted to determine the rule that applies to the request (step 1352). If the rule action is "ignore" (step 1354), then the literal name equals the virtual name (step 1356). If, however, the rule action is not "ignore" but is "redirect" (step 1358), then the literal name is determined from the virtual name as specified by the redirect rule (step 1360). If, however, the rule action is not "redirect" but is "isolate", then the literal name is determined from the virtual name using a scope-specific identifier (step 1362).

In some embodiments, the particular scope-specific identifier is specified in the rule. In other embodiments, the scope-specific identifier used is the one associated with the application isolation scope with which the requesting process is associated. This allows the window or window class to be used by any other applications associated with the same application isolation scope. In operating systems such as many of the Microsoft WINDOWS family of operating systems where window names and classes are already isolated within a session, this means that only applications executing in the same session that are associated with the same application isolation scope can use the window name or class.

In some of the family of Microsoft WINDOWS operating systems, the window name is used as the title of the window in the title bar. It is desirable to handle non-client area paint window message to ensure that the window title displayed in the window title bar reflects the virtual names and not the literal name for a particular window. When a non-client area paint message is intercepted, the virtual name associated with the window, if any, is retrieved from the mapping table. If a virtual name is retrieved, the non-client area is painted using the virtual name as the window title and it is indicated that the request message has been handled. If no virtual name is retrieved, the request is indicated as not handled, which passes the request on to the original function that paints the title bar, using the literal name of the window.

4.5 Out-of-process COM Server Virtualization

Software component technologies such as COM, CORBA, .NET and others allow software components to be developed, deployed, registered, discovered, activated or instantiated and utilized as discrete units. In most component models, components may execute in either the process of the caller or in a separate process on the same computer or on a separate computer entirely, although some components may only support a subset of these cases.

One or more unique identifiers identify these components. Typically the component infrastructure provides a service or daemon that brokers activation requests. A software process that wishes to begin using a component passes a request to the broker to activate the component specified by the component identifier. The broker activates the requested component, if possible, and returns a reference to the activated instance. In some of these component infrastructures, multiple versions of the same component may not co-exist because the component identifier remains the same from version to version.

Some of the members of the WINDOWS family of operating systems provide a component infrastructure called COM. COM components ("COM servers") are identified by a GUID called a Class Identifier (CLSID), and each component provides one or more interfaces each of which has its own unique interface identifier (UIID). The COM Service Control Manager (CSCM) is the broker for out-of-process activation requests and it provides interfaces that allow the caller to request activation of a COM server via CLSID. Although the following description will be phrased in terms of COM servers and COM clients, it will be understood by one of ordinary skill in the art that it applies to CORBA, .NET, and other software architectures that provide for dynamic activation of software components.

When COM components are installed onto a computer, they register their CLSIDs in a well-known portion of the registry database, along with the information needed by the CSCM to launch a new instance of the COM server.

For out of process COM servers, this may include the path and command line parameters to the executable to run. Multiple versions of the same COM server share the same CLSID, hence only one version can be installed onto a computer at a time.

In certain embodiments, an application (acting as a COM client) instantiates a COM server by calling a COM API (for example, CoCreateInstance() or CoCreateInstanceEx()). A parameter to this call specifies the desired activation context: in-process; out-of-process on the same computer; out-of-process on a remote computer; or allow the COM subsystem to determine which of these three cases to use. If it is determined that an out-of-process activation is required, the request including the CLSID is passed to the CSCM. The CSCM uses the registry database to locate the path and parameters needed to launch the executable that hosts the COM server. When that executable is launched, it registers all of the CLSIDs of all of the COM servers that it supports with the CSCM using the COM API CoRegisterClassObject(). If the requested CLSID is registered, the CSCM returns a reference to that COM server to the caller. All subsequent interaction between the COM client and the COM server takes place independently of the CSCM.

The isolation environment 200 previously described allows multiple instances of COM servers with the same CLSID to be installed on a computer, each in a different isolation scope (no more than one of which may be the system scope). However, this alone will not make those COM servers available to COM clients.

Fig. 14 depicts one embodiment of the steps to be taken to virtualize access to COM servers. In brief overview, a new CLSID, hereinafter called the Isolated CLSID (or ICLSID) is created for each out-of-process COM server that is launched into an isolation scope (step 1402). By definition this is a CLSID, and thus must be unique amongst all other CLSIDs, in other words it must have the properties of a GUID. A mapping table is created that maps the pair (CLSID,

application isolation scope) to ICLSID. A COM server registry entry is created for the ICLSID which describes how to launch the COM server, with launch parameters that start the COM server executable in the appropriate application isolation scope (step 1404). Calls by COM clients to COM APIs such as CoCreateInstance() and CoCreateInstanceEx() are hooked or intercepted (step 1406). If it is determined that (a) the request can be satisfied by an in-process COM server or (b) both the COM client and COM server are not associated with any isolation scope, then the request is passed unmodified to the original COM API and the result returned to the caller (step 1408). The appropriate instance of the COM server to use is identified (step 1410). If the selected COM server instance is in an application isolation environment its ICLSID is determined using the data structures outlined above. Otherwise, the CLSID in the request is used (step 1412). The original CoCreateInstance() or CoCreateInstanceEx() function is called, with the ICLSID if one was identified in step 1412. This passes the request to the CSCM (step 1414). The CSCM finds and launches the COM server executable in the normal fashion, by looking up the requested CLSID in the registry to determine launch parameters. If an ICLSID is requested, the ICLSID system scope registry entry as described in step 1404 is found and the COM server is launched in the appropriate application isolation scope (step 1416). The launched COM executable calls the hooked CoRegisterClassObject() API with the CLSIDs of the COM servers it supports and these are translated to the appropriate ICLSIDs which are passed to the original CoRegisterClassObject() API (step 1418). When the CSCM receives a response from a CoRegisterClassObject() call with the expected ICLSID, it returns a reference to that COM server instance to the caller (step 1420).

Still referring to FIG. 14, and in more detail, an ICLSID is created for each out-of-process COM server that is launched into an isolation scope (step 1402). In some embodiments, the ICLSID is created during installation of the COM server. In other embodiments, the ICLSID is created immediately after

installation. In still other embodiments, the ICLSID is created before the COM server is launched into the isolation scope. In all of these embodiments, the ICLSID may be created by hooking or intercepting the system calls that create or query the CLSID entry in the registry database. Alternatively, the ICLSID may be created by hooking or intercepting the COM API calls such as CoCreateInstance() and CoCreateInstanceEx() that create COM server instances. Alternatively, changes to the CLSID-specific portion of the registry database may be observed after an installation has taken place.

A mapping table is created that maps the pair (CLSID, application isolation scope) to ICLSID, along with the appropriate registry entries for a COM server with that ICLSID that describe how to launch the COM server, with launch parameters that start the COM server executable in the appropriate application isolation scope (step 1404). In many embodiments, this table is stored in a persistent memory element, such as a hard disk drive or a solid-state memory element. In other embodiments, the table may be stored in the registry, in a flat file, in a database or in a volatile memory element. In still other embodiments, the table may be distributed throughout the COM-specific portions of the registry database, for example by adding a new subkey specific to this purpose to each appropriate COM server entry identified by CLSID. Entries in this table may be created during or immediately after installation, by hooking or intercepting the calls that create the CLSID entry in the registry database, or by observing changes to the CLSID-specific portion of the registry database after an installation has taken place, or by hooking or intercepting the COM API calls such as CoCreateInstance() and CoCreateInstanceEx() that create COM server instances. Installation of a COM server into a specific isolation scope may be persistently recorded. Alternatively, the mapping of a particular COM server and isolation scope to ICLSID may be dynamically created and stored as an entry in a non-persistent database, or in the registry database.

Calls by COM clients to COM APIs, such as CoCreateInstance() and CoCreateInstanceEx(), are hooked or intercepted (step 1406). If it is determined that (a) the request can be satisfied by an in-process COM server or (b) both the COM client and COM server reside in the system scope (step 1407), then the request is passed unmodified to the original COM API and the result returned to the caller (step 1408).

If the request cannot be satisfied by an in-process COM server and either the COM client or the COM server do not reside in the system scope (step 1407), then the appropriate instance of the COM server to use is identified (step 1410). For embodiments in which COM clients execute in a particular isolation scope, preference may be given to COM servers installed into the same application isolation scope, followed by those installed into the system scope (possibly executing in the client's application isolation scope), followed by COM servers installed into other application isolation scopes. In some of these embodiments, COM servers installed into the system scope may execute in the same application isolation scope as the COM client. This may be controlled by the rules engine and administrative settings to allow this to happen for COM servers that execute correctly in this mode, and prevent it for COM servers that do not. For embodiments in which the COM client executes in the system scope, preference may be given to system scope COM servers followed by COM servers in isolation scopes. The COM client may specify a COM server to use in the call creating an instance of the COM server. Alternatively, a configuration store may store information identifying the COM server to be instantiated. In some embodiments, the specified COM server is hosted by another computer, which may be a separate, physical machine or a virtual machine. The mapping table described above in connection with step 1404 may be used to find the set of applicable COM servers and (if necessary) compute preference based on rules.

For embodiments in which the applicable COM server exists on another computer, a service or daemon that executes on the remote computer can be queried for the ICLSID to use. The COM client hook, if it determines that a remote COM server is required, first queries the service or daemon to determine the CLSID/ICLSID to use. The service or daemon determines an ICLSID corresponding to the CLSID given in the request. In some embodiments, the ICLSID returned by the service or daemon may be selected or created based on administrator-defined configuration data, rules contained in a rules engine, or built-in hard-coded logic. In other embodiments, the request may specify the isolation scope on the server to be used. In still other embodiments, the requested COM server may be associated with the server's system scope, in which case the CLSID associated with the COM server is returned. In still other embodiments, the requested COM server may be associated with one of the server's isolation scopes, in which case it returns the ICLSID associated with the instance of the COM server and the isolation scope. In some embodiments, a service or daemon as described above may be used to support launching local out-of-process COM servers.

If the selected COM server instance is in an application isolation environment on the local computer, its ICLSID is determined using the data structures described in connection with step 1404. If, instead, the selected COM server instance is in the system scope on the local computer, the CLSID in the request is used (step 1412). In some of these embodiments, an entry for the COM server using the ICLSID may be dynamically created.

If an ICLSID is returned, it is passed to the original COM API in place of the original CLSID. For example, the determined ICLSID may be passed to the original CoCreateInstance() or CoCreateInstanceEx() function, which passes the request to the CSCM (step 1414). For embodiments in which the COM server is hosted by another computer, the CSCM passes the ICLSID to the computer

hosting the COM server, where that computer's CSCM handles the COM server launch.

The CSCM finds and launches the COM server executable in the normal fashion, by looking up the requested CLSID or ICLSID in the registry to determine launch parameters. If an ICLSID is requested, the ICLSID system scope registry entry as described in step 1404 is found and the COM server launched in the appropriate application isolation scope (step 1416).

If the launched COM server instance executes in an application isolation scope (whether installed into that scope or installed into the system scope), the COM API function CoRegisterClassObject()of the COM server instance is hooked or intercepted. Each CLSID that is passed to CoRegisterClassObject() is mapped to the corresponding ICLSID, using the mapping table as defined in step 1404. The original CoRegisterClassObject() API is called with the ICLSID (step 1418).

When the CSCM receives a response from a CoRegisterClassObject() call with the expected ICLSID, it returns a reference to that COM server instance to the caller (step 1420).

This technique supports COM server execution when the COM client and COM server execute in any combination of application isolation scopes (including different scopes) and the system scope. The ICLSID is specific to the combination of server (identified by CLSID) and the desired appropriate isolation scope. The client need only determine the correct ICLSID (or the original CLSID if the server is in installed into and executing in the system scope).

4.6 Virtualized File Type Association (FTA)

File type association is a well-known graphical user interface technique for invoking execution of application programs. A user is presented with a graphical icon representing a data file. The user selects the data file using keyboard commands or using a pointing device, such as a mouse, and clicks, or double-clicks, on the icon to indicate that the user would like to open the file. Alternately,

in some computing environments, the user enters the path to the file at a command line prompt in place of a command. The file typically has an associated file type indication which is used to determine an application program to use when opening the file. This is generally done using a table that maps the file type indication to a specific application. In many members of the family of Microsoft WINDOWS operating systems, the mapping is typically stored in the registry database in a tuple including the file type indicator and the full pathname identifying the application to be executed, and only one application program may be associated with any particular file type.

In the described isolation environment, multiple versions of an application may be installed and executed on a single computer. Thus, in these environments, the relationship between file type and associated application program is no longer a one-to-one relationship but is, instead, a one-to-many relationship. A similar problem exists for MIME attachment types. In these environments, this problem is solved by replacing the pathname identifying the application program to be launched when a given file type is selected. The pathname is replaced with that of a chooser tool that gives to the user a choice of application programs to launch.

Referring now to FIG. 15, and in brief overview, a request to write file type association data to a configuration store is intercepted (step 1502). A determination is made whether the request is updating the file type association information in the configuration store (step 1504). If not, i.e., if the entry already exists, no update occurs (step 1506). Otherwise, a new entry is created using the virtualization techniques described above in sections 4.1.4 or 4.2.4, or the existing entry is updated (step 1508). The new or updated entry, which is virtualized for the appropriate isolation scope, maps the file type to a chooser tool which allows the user to select which of multiple application programs to use when viewing or editing the file.

Still referring to FIG. 15, and in more detail, a request to write file-type association data to a configuration store is intercepted (step 1502). In some embodiments, the configuration store is the WINDOWS registry database. The request to write data to the configuration store may be intercepted by a user mode hooking function, a kernel mode hooking function, a file system filter driver, or a mini-driver.

A determination is made whether the request seeks to update file-type-association information in the configuration store (step 1504). In one embodiment this is accomplished by detecting if the intercepted request indicates that it intends to modify the configuration store. In another embodiment, the target of the request is compared to the information included in the request to determine if the request is attempting to modify the configuration store. For embodiments in which the configuration store is a registry database, the request to modify the registry is intercepted, as described above in Section 4.2.

If it is determined that the request is not attempting to update the configuration store, no update occurs (step 1506). In some embodiments, it is determined that no attempt to update the configuration store is made because the intercepted request is a read request. In other embodiments, this determination may made when the target entry in the configuration store and the information included in the intercepted request are identical, or substantially identical.

If, however, it is determined in step 1504 that the request intends to update the configuration store, then a new entry is created in the configuration store, or the existing entry is updated (step 1508). In some embodiments, rules determine which isolation scope the entry is created or updated in. In some embodiments, the new entry is created or the existing entry is updated in the system scope or the application isolation scope. In many embodiments, the new entry is created or the existing entry is updated in the appropriate user isolation scope. If a new entry is created, then it, rather than identifying the application

program identified in the intercepted request, lists a chooser application as the application to be used when a file of a particular type is accessed. In some embodiments, the chooser tool is updated automatically when a new version of an application program is installed, or when another application that handles the same file type is installed, or when an application registers or deregisters itself to handle files of that particular type. In some embodiments, the chooser tool can incorporate into its list of suitable applications any applications registered to handle the same file type in the portion of the configuration store maintained in other scopes, such as the system scope, and the application scope if the chooser tool executes in the user scope. If an existing entry is updated, and the existing entry already lists the chooser application as the application to be used when a file of that particular file type is used, then the list of applications presented by the chooser for that file type may be updated to include the updating application. If the existing entry is updated, but it does not list the chooser application, then the updated entry is made to list the chooser application as the application to be used when a file of that particular file type is used. In these embodiments, the information relating to the associated applications may be stored in an associated configuration file or, in some embodiments, as an entry in the registry database.

The chooser application may present to the user a list of applications associated with the selected file type. The application may also allow the user to choose the application program that the user would like to use to process the file. The chooser then launches the application program in the appropriate scope: system scope; application isolation scope; or user isolation scope. In some embodiments, the chooser tool maintains the identity of the default application program associated with a file type. In these embodiments, the default application may be used by processes that do not have access to the desktop or are configured to use the default handler without presenting the user with a choice.

4.7 Dynamic movement of processes between isolation environments

An additional aspect of the invention is the facility to move a running process between different virtual scopes. In other words, the aggregated view of native resources presented to the application instance by the isolation environment 200 may be changed to a different aggregated view while the application is executing. This allows processes that have been isolated within a particular isolation scope to be "moved" to another isolation scope while the process is running. This is particularly useful for system services or processes of which only one instance may execute at a time, such as the MSI service in WINDOWS operating systems. This aspect of the invention may also be used to allow a user to work in several isolation scopes sequentially.

Referring to FIG. 16, and in brief overview, one embodiment of a process for moving processes between one isolation scope and a second isolation scope, or between the system scope and an isolation scope, is shown. As used in this description, the term "target isolation scope" will be used to refer to the isolation scope, including the system scope, to which the processes is being moved and the term "source isolation scope" will be used to refer to the isolation scope, including the system scope, from which the process is being moved. As shown in FIG. 16, and in brief overview, a method for moving a process to a target isolation scope includes the steps of: ensuring that the process is in a safe state (step 1602); changing the association of the process from its source isolation scope to the target isolation scope in the rules engine (step 1604); changing the association of the process from the source isolation scope to the target isolation scope for any filter driver or hooks (step 1606); and allowing the process to resume execution (step 1608).

Still referring to FIG. 16, and in more detail, the process should be in a "safe" state while being moved to a different isolation scope (step 1602). In some embodiments, the process is monitored to determine when it is not processing requests. In these embodiments, the process is considered to be in a

"safe" state for moving when no requests are processed by the process. In some of these embodiments, once the process is considered to be in a "safe" state, new requests to the process are delayed until the process is moved. In other embodiments, such as in connection with diagnostic applications, a user interface may be provided to trigger the change in isolation scope. In these embodiments, the user interface may run code that puts the process to be moved into a "safe" state. In still other embodiments, an administration program may force the process into a "safe" state by delaying all incoming requests to the process and waiting for the process to complete execution of any active requests.

The rules associated with the target isolation scope are loaded into the rules engine if they do not already exist in the rules engine (step 1603).

The association of the process with a source isolation scope is changed in the rules engine (step 1604). As described above, a process can be associated with any isolation scope. That association is used by the rules engine on every request for a virtual native resource to determine the rule to apply to the request. The application instance can be associated with a target isolation scope by changing the appropriate data structures in the rules engine. In some embodiments, a new database entry is written associating the process with a new isolation scope. In other embodiments, a tree node storing an identifier for the isolation scope with which the process is associated is overwritten to identify the new isolation scope. In still other embodiments, an operating system request can made to allocate additional storage for a process to store the rules associated with the target isolation scope or, in some embodiments, an identifier of the rules.

The association of the process with the source isolation scope is changed wherever the association or the rules are stored outside of the rules engine, such as filter drivers, kernel mode hooks, or user mode hooks (step 1606). For embodiments in which the association between a process and isolation scope

rules is maintained based on PID, the association between the processes PID and the rule set is changed. For embodiments in which a PID is not used to maintain the association between a process and the applicable set of isolation rules, the user mode hooking function may be altered to access the rule set associated with the target isolation scope. For embodiments in which process associations with rule sets for isolation scopes are maintained in a rule engine, it is sufficient to change the association stored in the rule engine in step 1604 above.

The process is allowed to resume execution in the new isolation scope (step 1610). For embodiments in which new requests were delayed or prohibited from being made, those requests are issued to the process and new requests are allowed.

In one particularly useful aspect, the method described above may be used to virtualize MSI, an installation packaging and installation technology produced by Microsoft and available in some of the Microsoft WINDOWS family of operating systems. An application packaged by this technology for installation is called an MSI package. Operating systems which support this technology have a WINDOWS service called the MSI service which assists in installing MSI packages. There is a single instance of this service on the system. Processes that wish to install MSI packages run an MSI process in their session which makes COM calls to the MSI service.

MSI installations can be virtualized to install MSI packages into an application isolation environment. Conceptually, this can be achieved by hooking or intercepting the calls made to the MSI API in the installation session to the MSI service. A mutex can be used to ensure that only one installation takes place at a time. When a call to the MSI API requesting to start a new installation is received or intercepted, and the calling process is associated with a particular application isolation scope, the MSI service is placed into the context of that isolation scope before the call is allowed to proceed. The installation proceeds

as the MSI service performs its normal installation actions, although native resource requests by the MSI service are virtualized according to the applicable isolation scope. When the end of the installation process is detected, the association between the MSI service and the isolation scope is removed. Although described above with respect to MSI, the technique described is applicable to other installation technologies.

EQUIVALENTS

The present invention may be provided as one or more computer-readable programs embodied on or in one or more articles of manufacture. The article of manufacture may be a floppy disk, a hard disk, a CD-ROM, a flash memory card, a PROM, a RAM, a ROM, or a magnetic tape. In general, the computer-readable programs may be implemented in any programming language, LISP, PERL, C, C++, PROLOG, or any byte code language such as JAVA. The software programs may be stored on or in one or more articles of manufacture as object code.

Having described certain embodiments of the invention, it will now become apparent to one of skill in the art that other embodiments incorporating the concepts of the invention may be used. Therefore, the invention should not be limited to certain embodiments, but rather should be limited only by the spirit and scope of the following claims.

CLAIMS

What is claimed is:

1. A method for moving an executing process from a first isolation scope to a second isolation scope, the method comprising the steps of:

   (a) determining that a process is in a state suitable for moving;

   (b) changing an association of the process from a first isolation scope to a second isolation scope; and

   (c) loading at least one rule associated with the second isolation scope.

2. The method of claim 1 wherein step (a) comprises determining that a process is in a state suitable for moving by monitoring whether the process is processing a request.

3. The method of claim 2 wherein step (a) comprises putting a process in a state suitable for moving.

4. The method of claim 3 further comprising putting the process in the state suitable for moving via one of a user interface and an administration program.

5. The method of claim 3 further comprising the step of prohibiting new requests to the process.

6. The method of claim 3 further comprising the step of queuing requests to the process.

7. The method of claim 6 further comprising the step of processing the queued requests after associating the process with the second isolation scope.

8. The method of claim 1 wherein step (b) comprises writing to a rules engine information associating the process with a second isolation scope.

9. The method of claim 1 further comprising the step of changing an association of the process from the first isolation scope to the second isolation scope in a file system filter driver.

10. The method of claim 1 further comprising the step of changing an association of the process from the first isolation scope to the second isolation scope in one of a kernel hooking function and a user mode hooking function.

11. The method of claim 1 further comprising the step of changing an association of the process from the first isolation scope to the second isolation scope in a mini-filter.

12. The method of claim 1 further comprising the step of suspending execution by the process.

13. The method of claim 12 further comprising resuming execution by the process.

14. A method for moving an executing process into an isolation scope, the method comprising the steps of:

(a) determining that a process is in a state suitable for moving;
(b) associating the process with an isolation scope; and
(c) loading at least one rule associated with the isolation scope.

15. The method of claim 14 wherein step (a) comprises determining that a process is in a state suitable for moving by monitoring whether the process is processing a request.

16. The method of claim 15 wherein step (a) comprises putting a process in a state suitable for moving.

17. The method of claim 16 further comprising putting the process in the state suitable for moving via one of a user interface and an administration program.

18. The method of claim 16 further comprising the step of prohibiting new requests to the process.

19. The method of claim 16 further comprising the step of queuing requests to the process.

20. The method of claim 19 further comprising the step of processing the queued requests after associating the process with the isolation scope.

21. The method of claim 14 wherein step (b) comprises writing to a rules engine information associating the process with an isolation scope.

22. The method of claim 14 further comprising the step of associating the process to the isolation scope in a file system filter driver.

23. The method of claim 14 further comprising the step of associating the process to the isolation scope in one of a kernel hooking function and user mode hooking function.

24. The method of claim 14 further comprising the step of associating the process to the isolation scope in a mini-filter.

25. The method of claim 14 further comprising the step of suspending execution by the process.

26. The method of claim 25 further comprising resuming execution by the process.

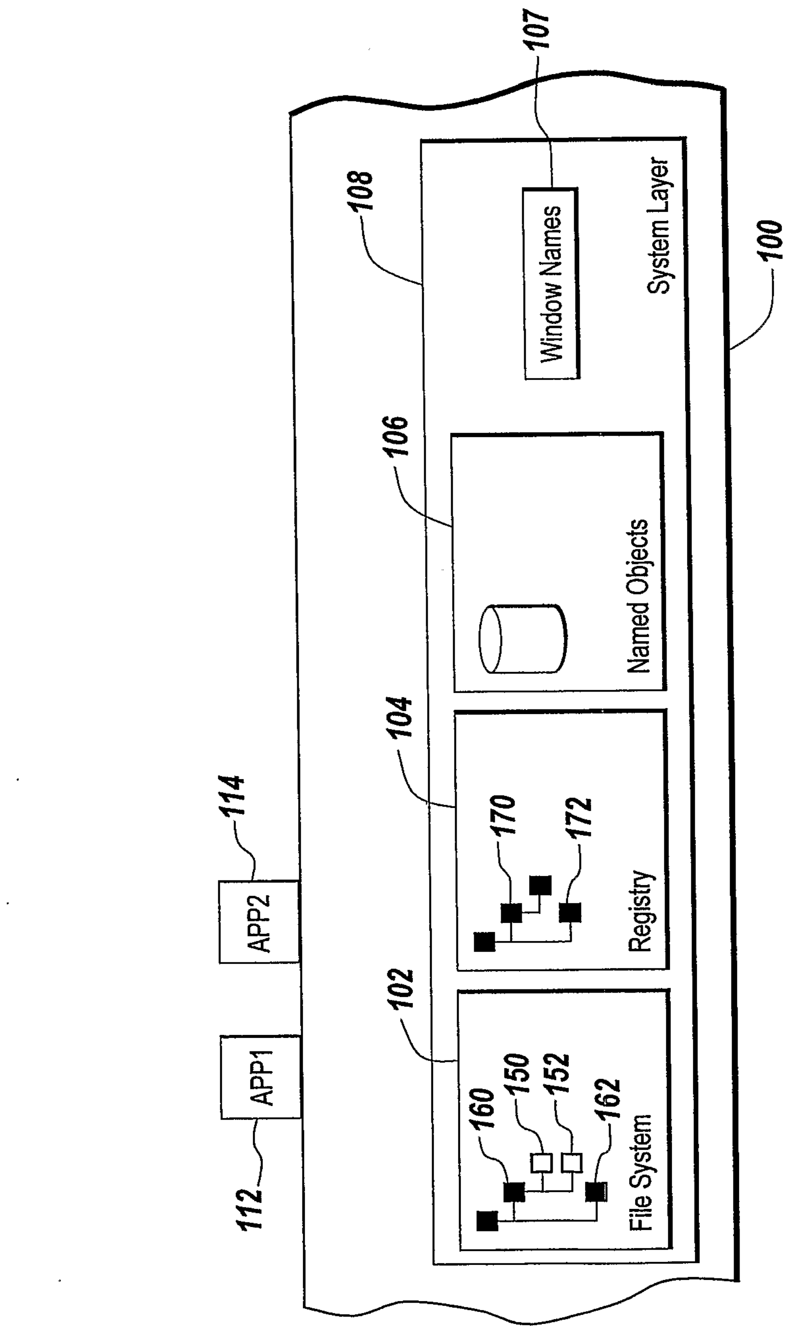


Fig. 1A
(Prior Art)

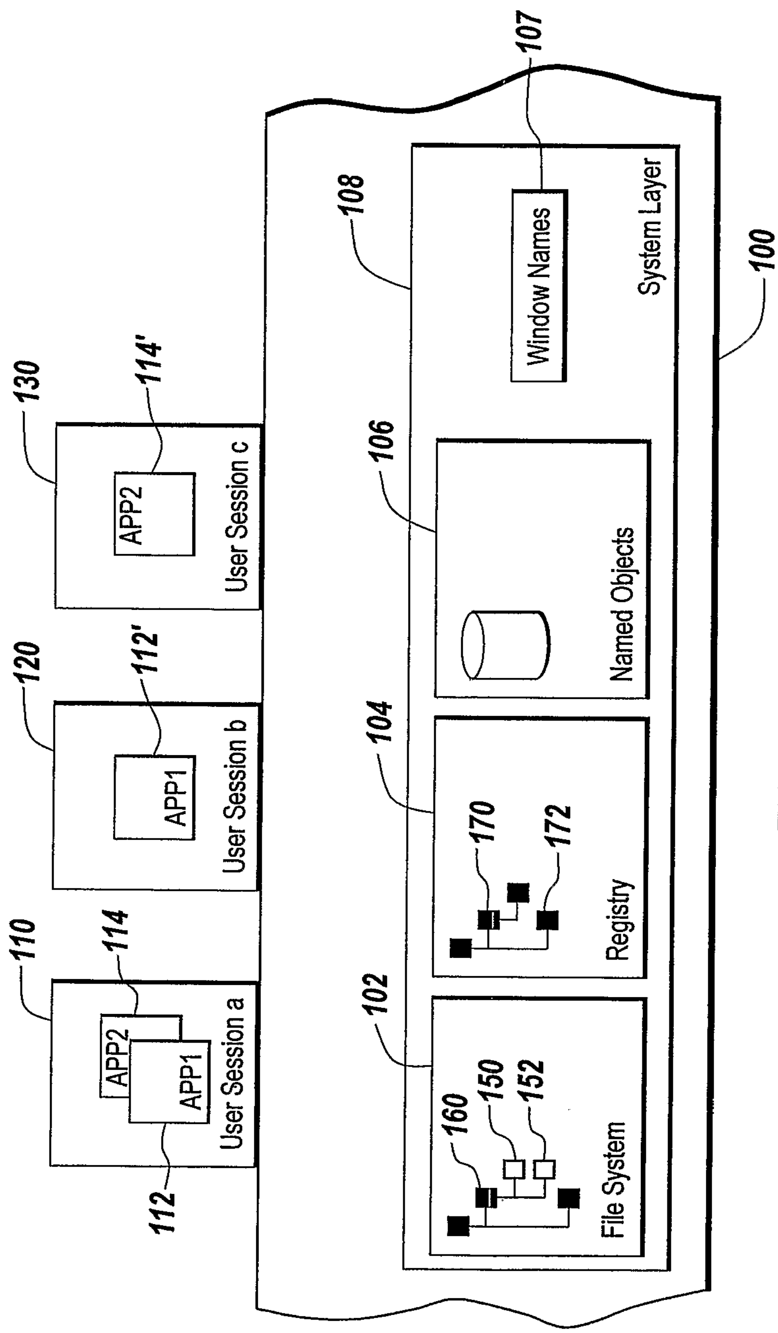


Fig. 1B
(Prior Art)

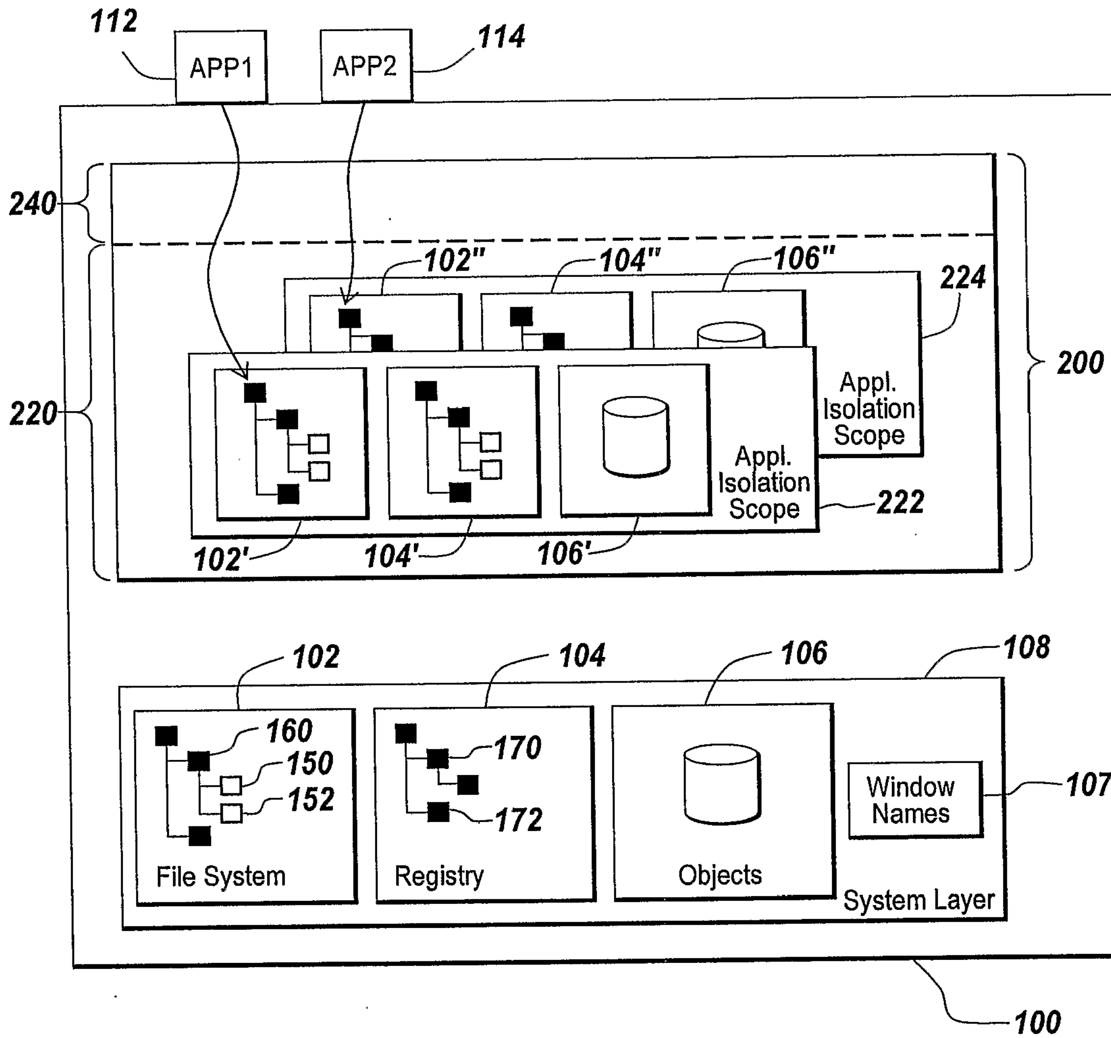

112
APP1
APP2
114
240
220
102"
104"
106"
224
200
Appl.
Isolation
Scope
Appl.
Isolation
Scope
222
102'
104'
106'
102
104
106
108
160
150
152
170
172
File System
Registry
Objects
Window
Names
107
System Layer
100

Fig. 2A

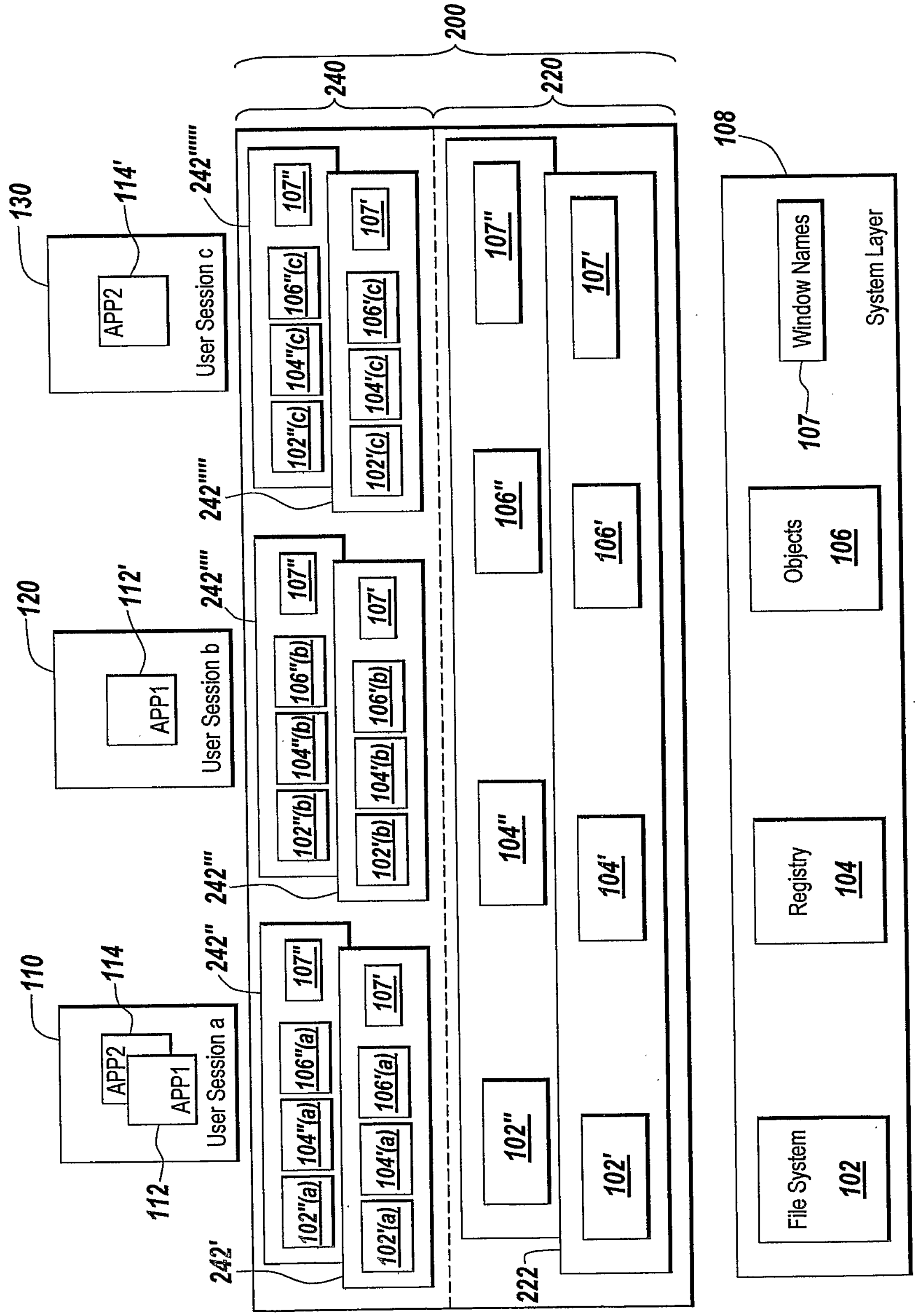

200
240
220
108
130
114'
242'''''
APP2
User Session c
120
112'
APP1
User Session b
110
114
APP2
APP1
User Session a
112
242'
242''
242'''
242''''
222
107''
107'
106''(c)
104''(c)
102''(c)
106'(c)
104'(c)
102'(c)
106''(b)
104''(b)
102''(b)
106'(b)
104'(b)
102'(b)
106''(a)
104''(a)
102''(a)
106'(a)
104'(a)
102'(a)
106''
106'
104''
104'
102''
102'
Window Names
107
System Layer
Objects
106
Registry
104
File System
102

Fig. 2B

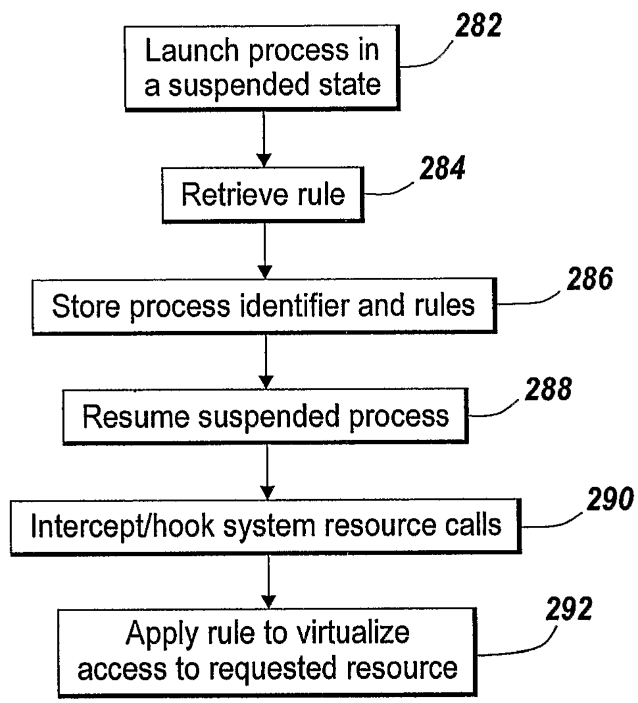

Launch process in a suspended state
282
Retrieve rule
284
Store process identifier and rules
286
Resume suspended process
288
Intercept/hook system resource calls
290
Apply rule to virtualize access to requested resource
292

Fig. 2C

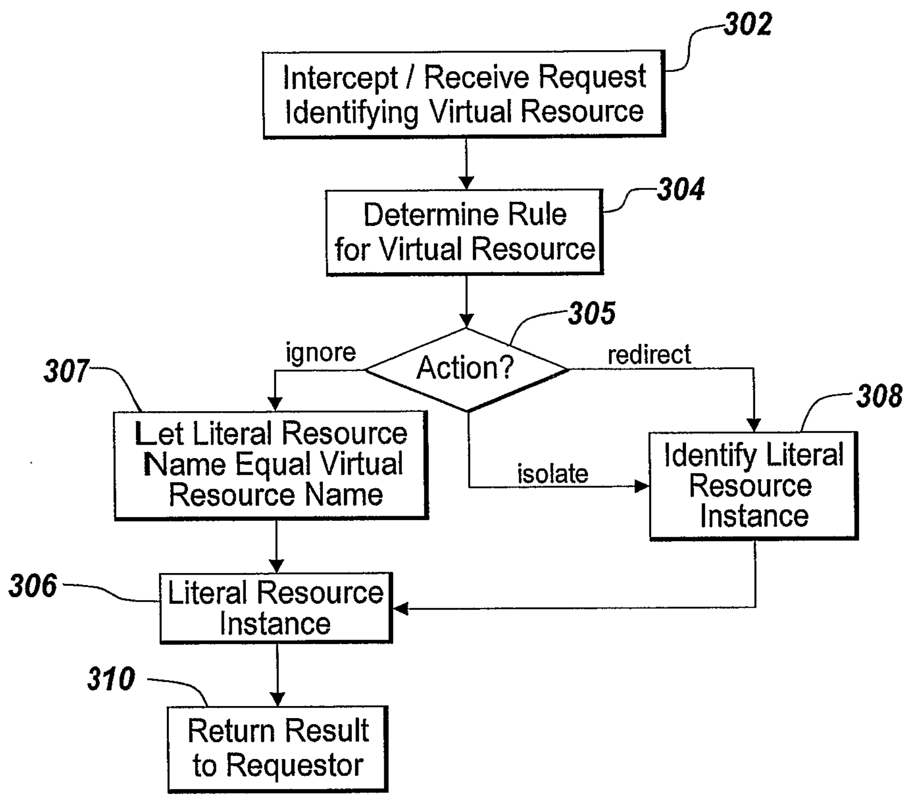

302
Intercept / Receive Request Identifying Virtual Resource
304
Determine Rule for Virtual Resource
305
Action?
ignore
redirect
isolate
307
Let Literal Resource Name Equal Virtual Resource Name
308
Identify Literal Resource Instance
306
Literal Resource Instance
310
Return Result to Requestor

Fig. 3A

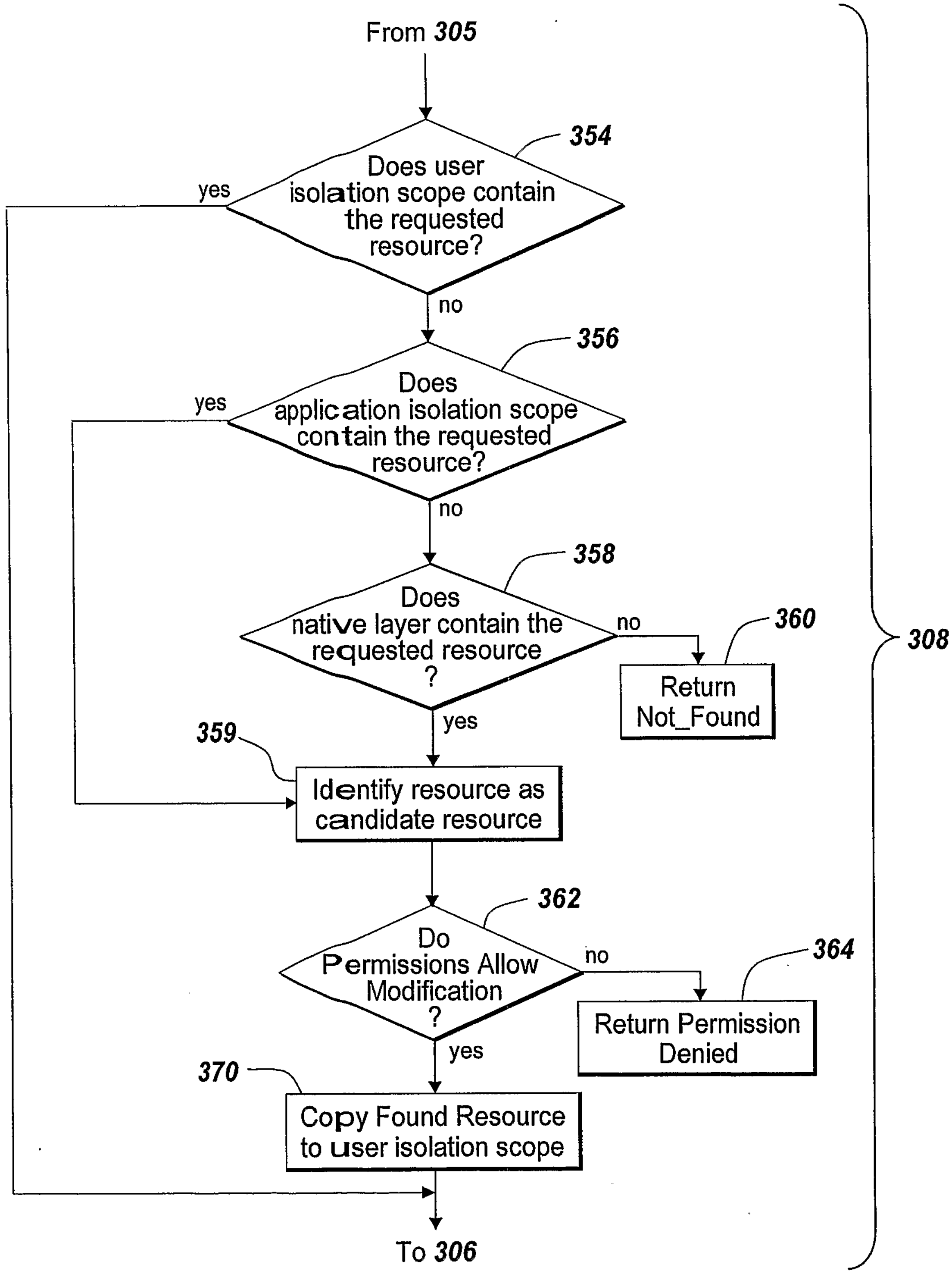

From 305
354
Does user isolation scope contain the requested resource?
yes
no
356
Does application isolation scope contain the requested resource?
yes
no
358
Does native layer contain the requested resource ?
no
360
Return Not_Found
yes
359
Identify resource as candidate resource
362
Do Permissions Allow Modification ?
no
364
Return Permission Denied
yes
370
Copy Found Resource to user isolation scope
To 306
308

Fig. 3B

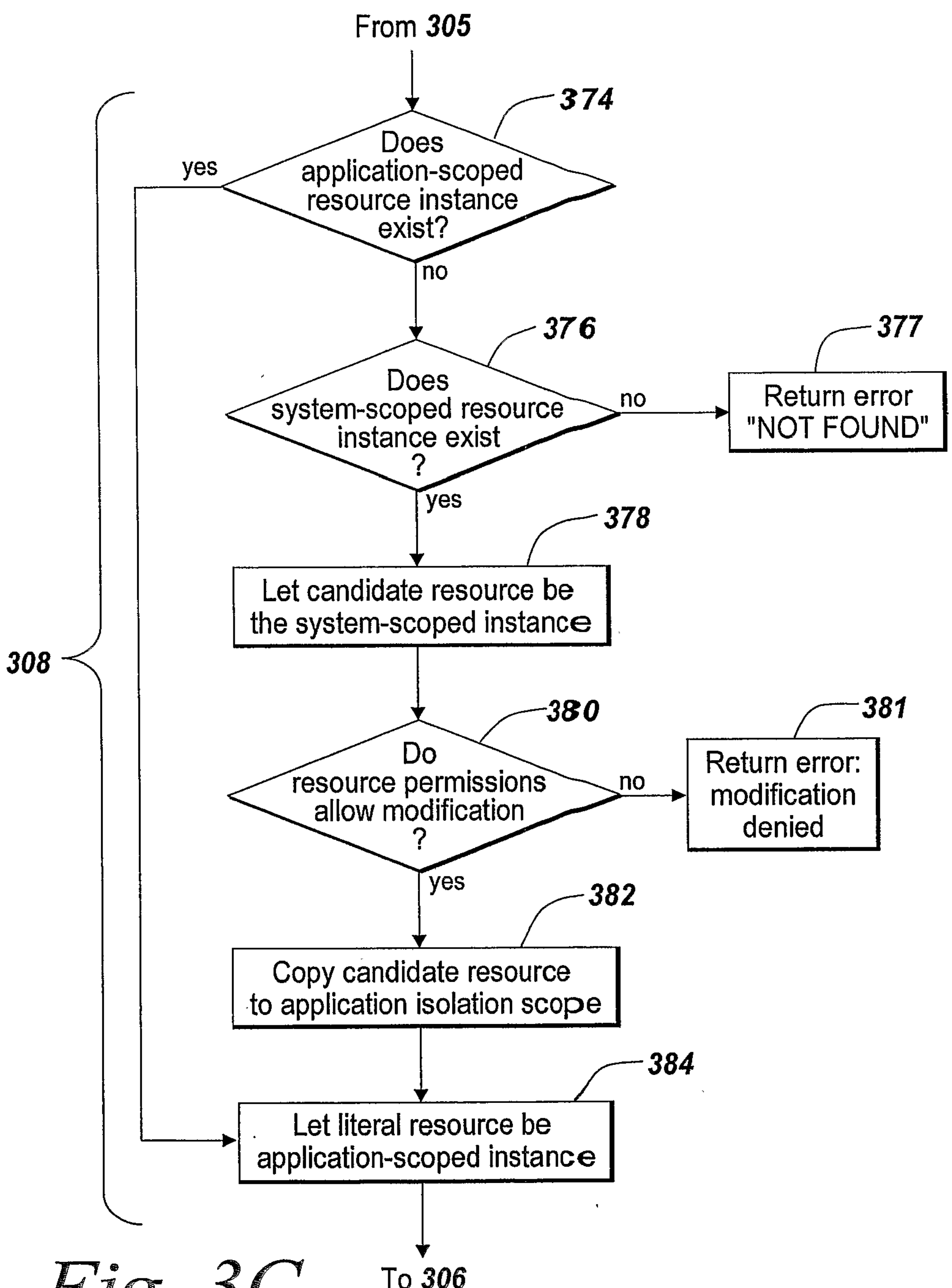

From 305
374
Does application-scoped resource instance exist?
yes
no
376
Does system-scoped resource instance exist ?
no
377
Return error "NOT FOUND"
yes
378
Let candidate resource be the system-scoped instance
308
380
Do resource permissions allow modification ?
no
381
Return error: modification denied
yes
382
Copy candidate resource to application isolation scope
384
Let literal resource be application-scoped instance
To 306

Fig. 3C

9/24

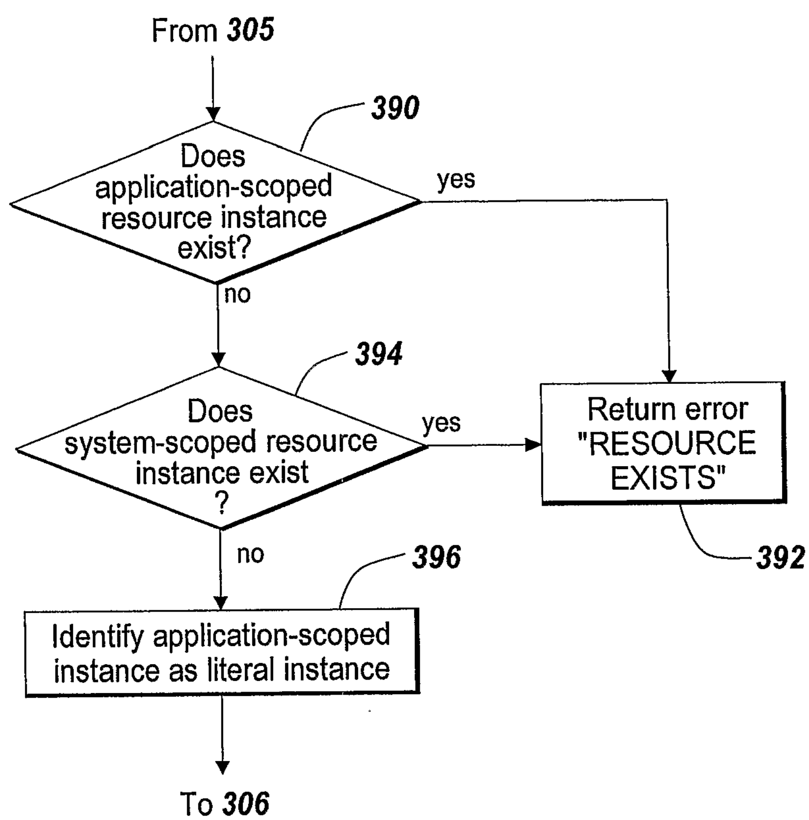
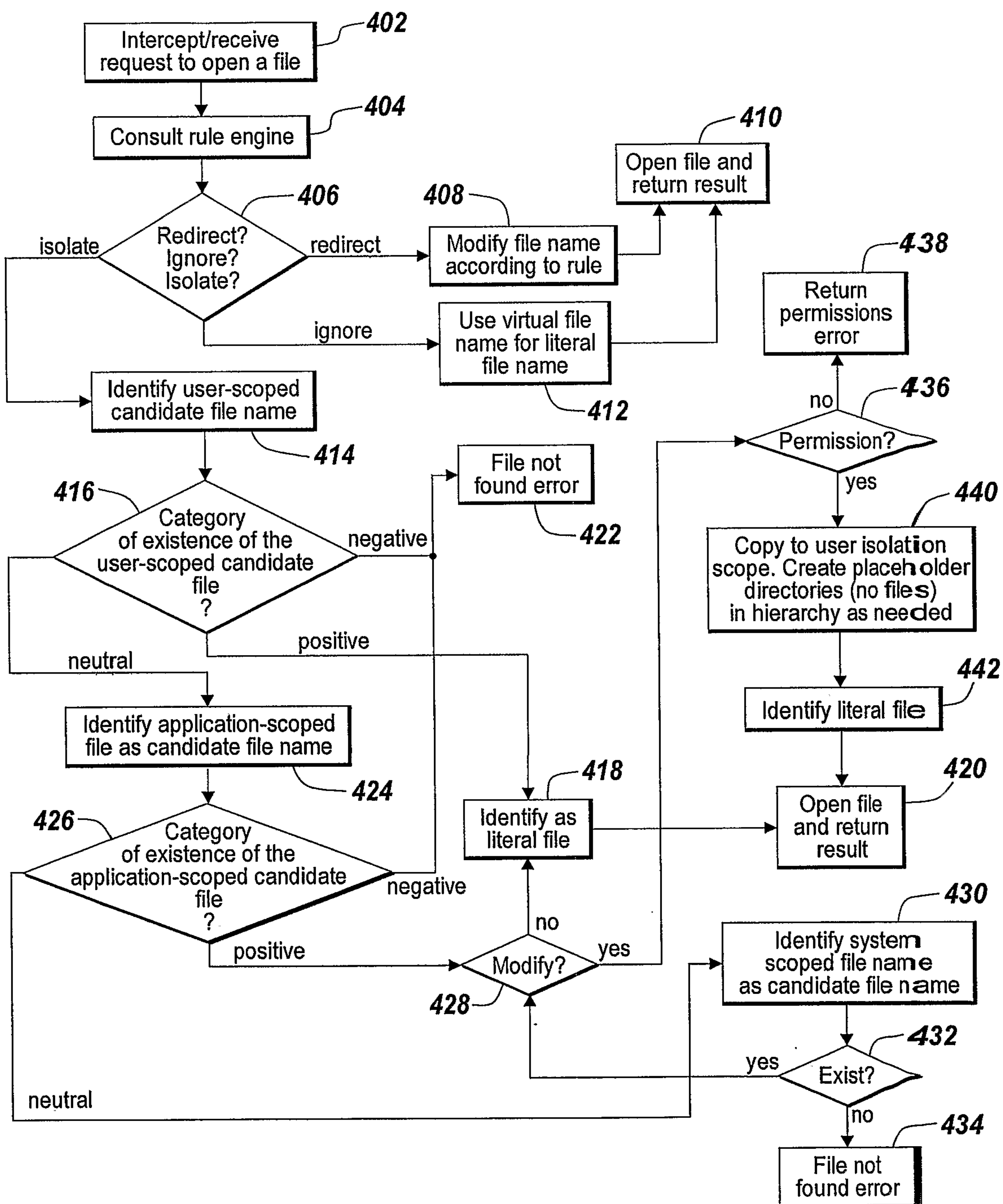

From 305
390
Does application-scoped resource instance exist?
yes
no
394
Does system-scoped resource instance exist ?
yes
Return error "RESOURCE EXISTS"
392
no
396
Identify application-scoped instance as literal instance
To 306

Fig. 3D



Intercept/receive request to open a file
402
Consult rule engine
404
Redirect? Ignore? Isolate?
406
isolate
redirect
ignore
Modify file name according to rule
408
Open file and return result
410
Use virtual file name for literal file name
412
Identify user-scoped candidate file name
414
Return permissions error
438
Permission?
436
no
yes
Copy to user isolation scope. Create placeholder directories (no files) in hierarchy as needed
440
File not found error
422
Category of existence of the user-scoped candidate file ?
416
negative
positive
neutral
Identify literal file
442
Identify application-scoped file as candidate file name
424
Identify as literal file
418
Open file and return result
420
Category of existence of the application-scoped candidate file ?
426
negative
positive
neutral
Modify?
428
no
yes
Identify system scoped file name as candidate file name
430
Exist?
432
yes
no
File not found error
434

Fig. 4

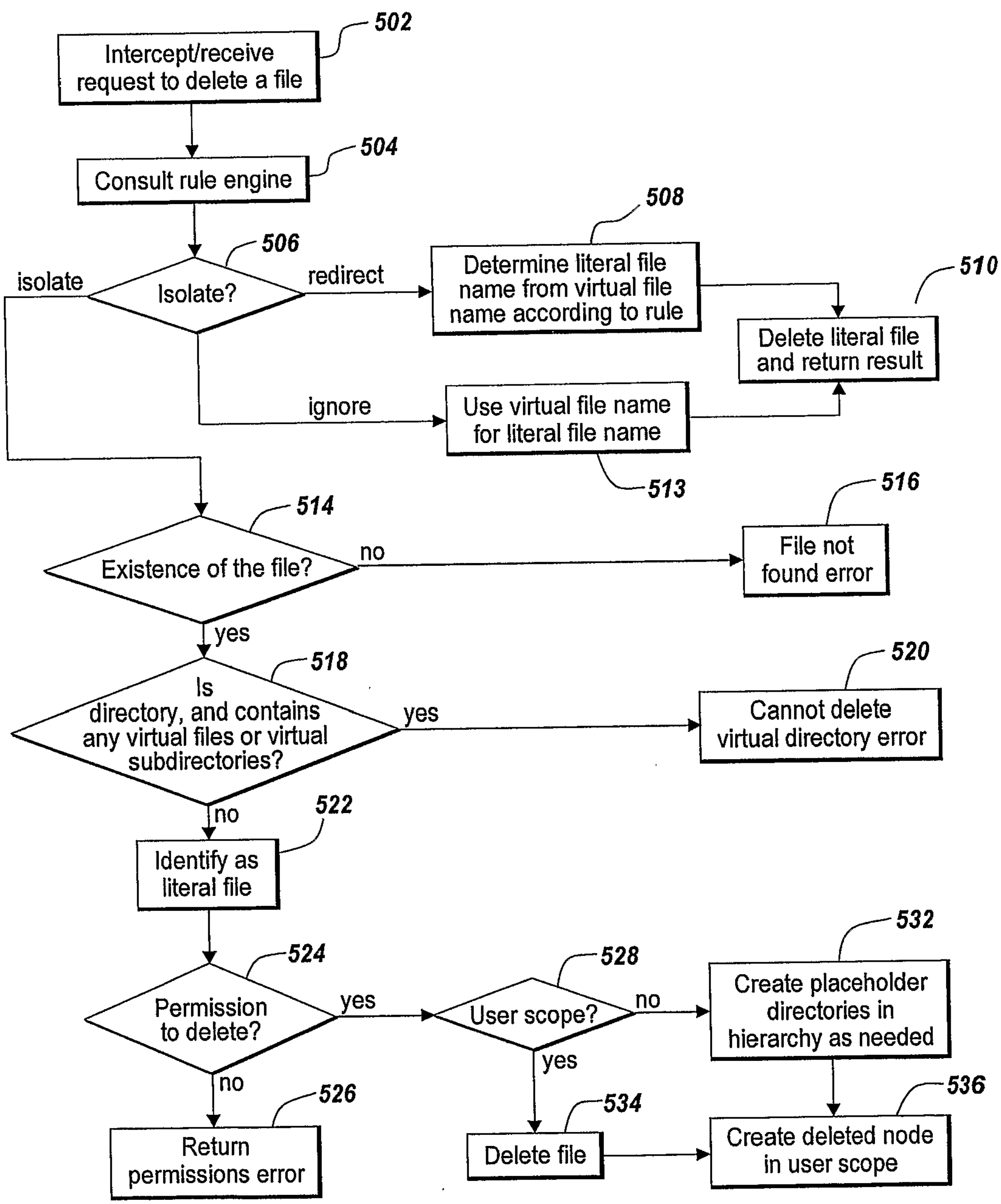

Intercept/receive request to delete a file
502
Consult rule engine
504
Isolate?
506
isolate
redirect
ignore
Determine literal file name from virtual file name according to rule
508
Delete literal file and return result
510
Use virtual file name for literal file name
513
Existence of the file?
514
no
yes
File not found error
516
Is directory, and contains any virtual files or virtual subdirectories?
518
yes
no
Cannot delete virtual directory error
520
Identify as literal file
522
Permission to delete?
524
yes
no
Return permissions error
526
User scope?
528
no
yes
Create placeholder directories in hierarchy as needed
532
Delete file
534
Create deleted node in user scope
536

Fig. 5

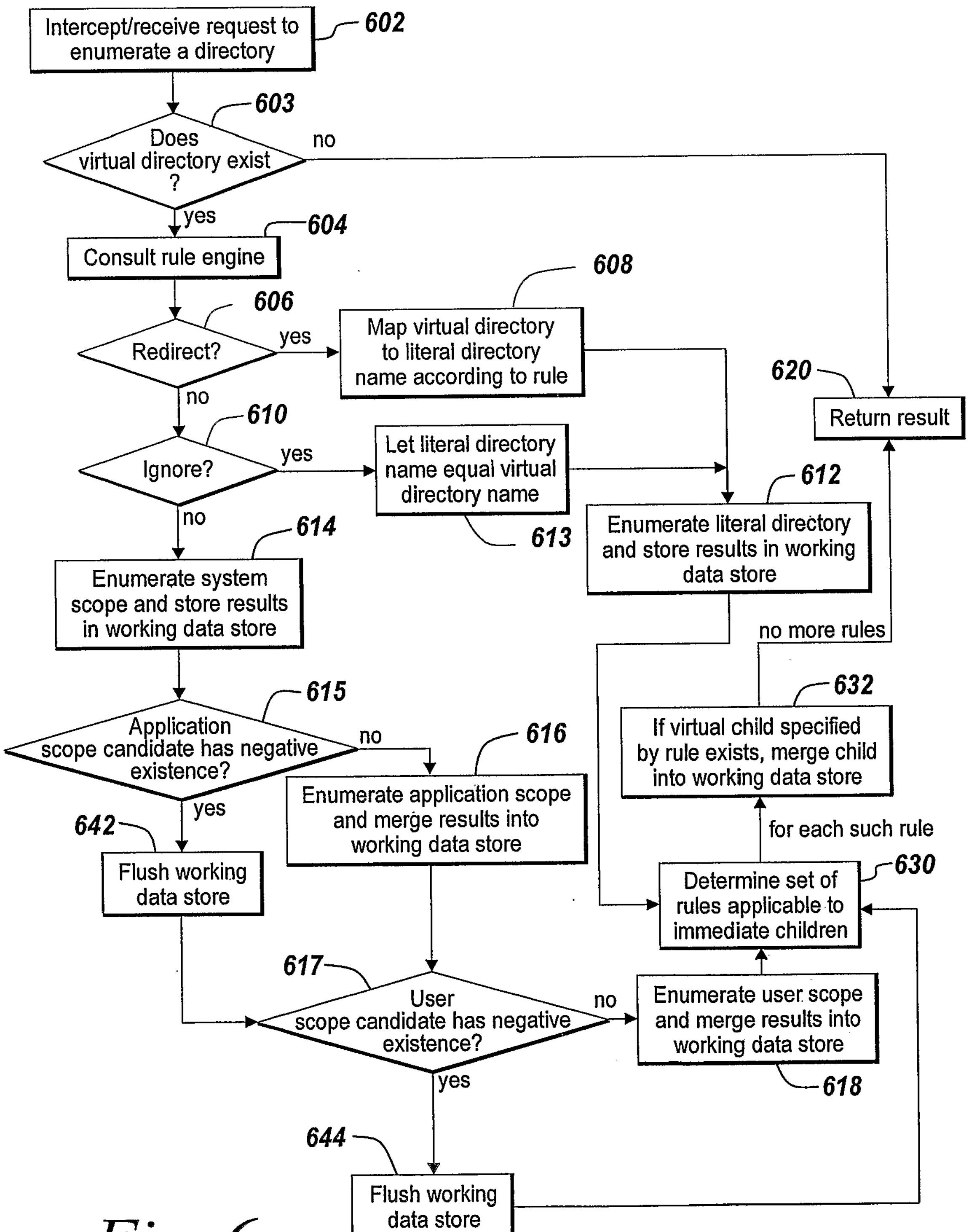

Intercept/receive request to enumerate a directory
602
Does virtual directory exist ?
603
no
yes
Consult rule engine
604
Redirect?
606
yes
Map virtual directory to literal directory name according to rule
608
no
Ignore?
610
yes
Let literal directory name equal virtual directory name
613
no
Enumerate system scope and store results in working data store
614
Return result
620
Enumerate literal directory and store results in working data store
612
no more rules
If virtual child specified by rule exists, merge child into working data store
632
for each such rule
Determine set of rules applicable to immediate children
630
Application scope candidate has negative existence?
615
no
Enumerate application scope and merge results into working data store
616
yes
Flush working data store
642
User scope candidate has negative existence?
617
no
Enumerate user scope and merge results into working data store
618
yes
Flush working data store
644

Fig. 6

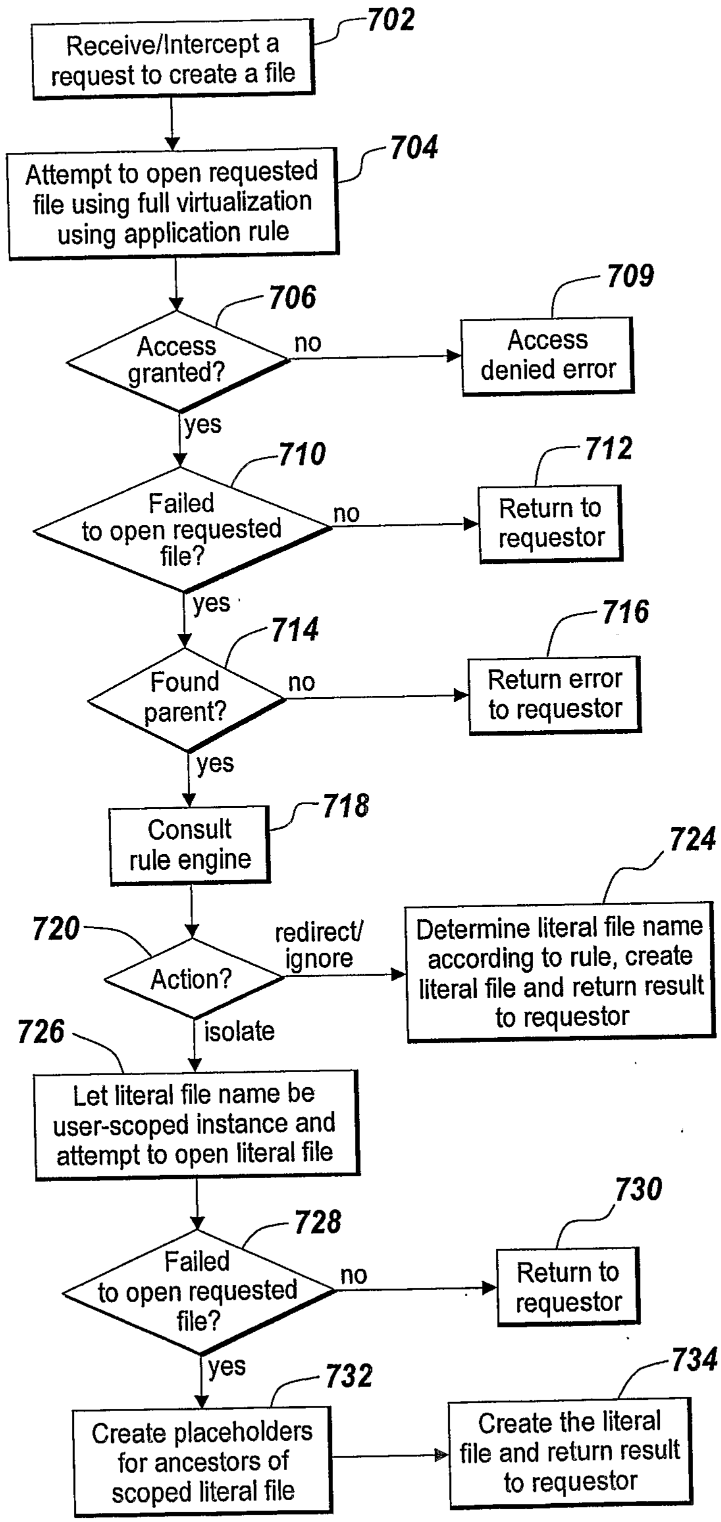

Receive/Intercept a request to create a file
702
Attempt to open requested file using full virtualization using application rule
704
Access granted?
706
no
Access denied error
709
yes
Failed to open requested file?
710
no
Return to requestor
712
yes
Found parent?
714
no
Return error to requestor
716
yes
Consult rule engine
718
Action?
720
redirect/ ignore
Determine literal file name according to rule, create literal file and return result to requestor
724
isolate
726
Let literal file name be user-scoped instance and attempt to open literal file
Failed to open requested file?
728
no
Return to requestor
730
yes
Create placeholders for ancestors of scoped literal file
732
Create the literal file and return result to requestor
734

Fig. 7

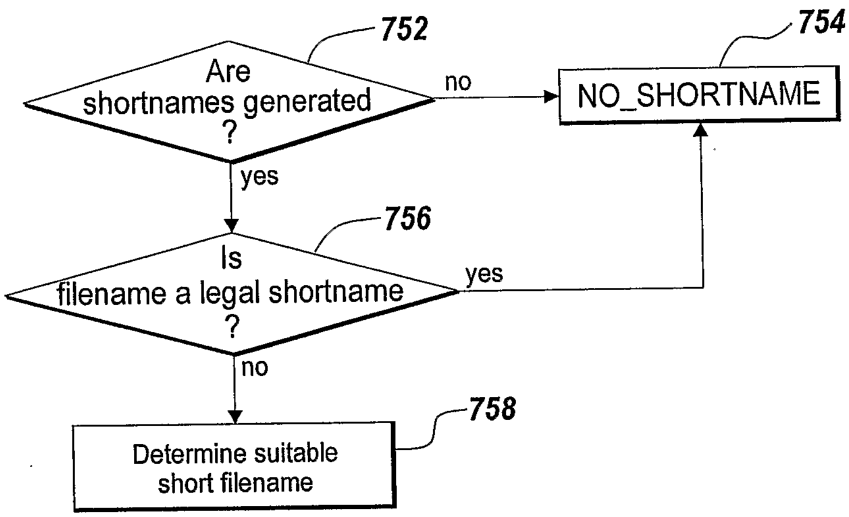

752
Are shortnames generated ?
no
754
NO_SHORTNAME
yes
756
Is filename a legal shortname ?
yes
no
758
Determine suitable short filename

Fig. 7A

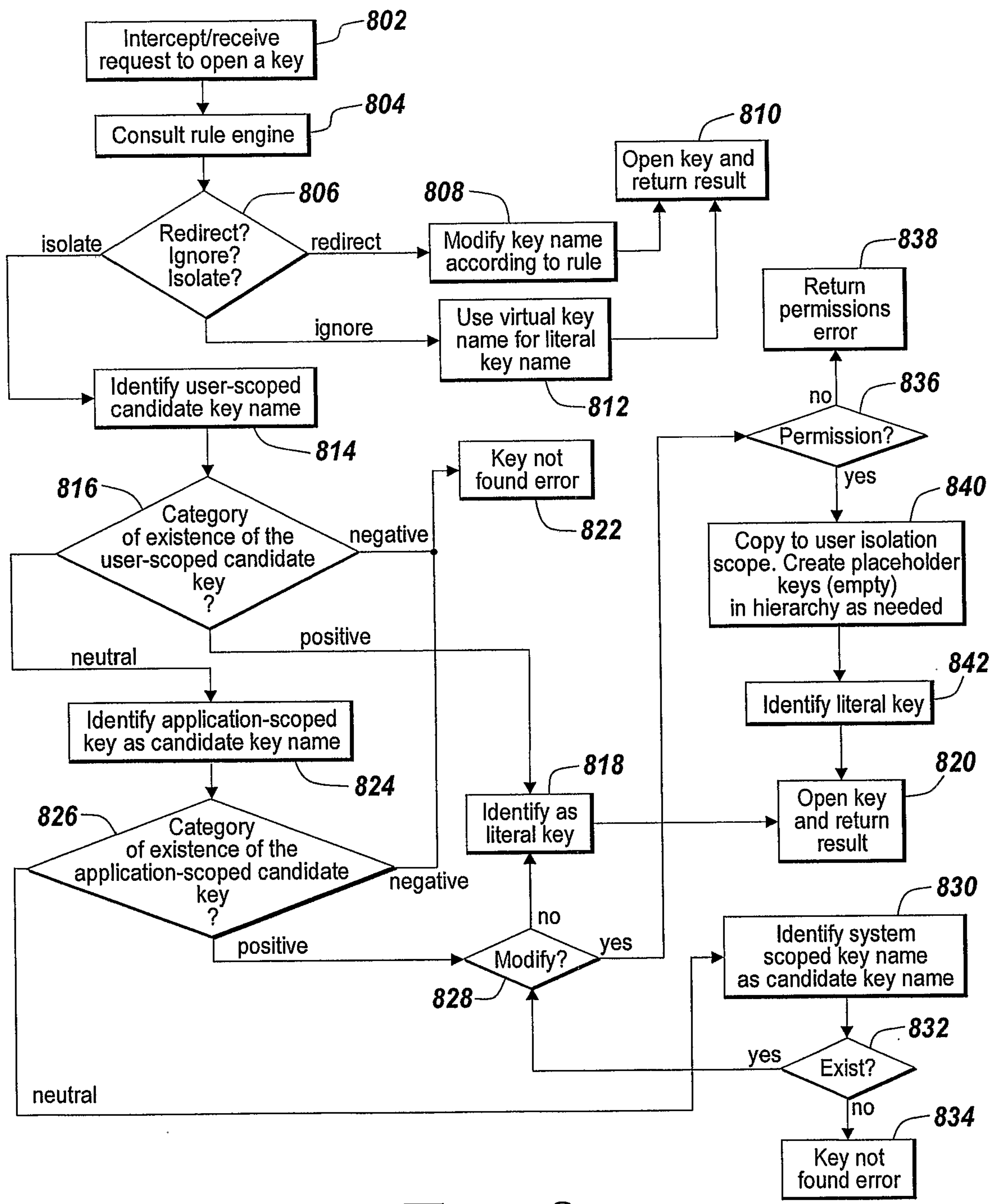

Intercept/receive request to open a key
802
Consult rule engine
804
Redirect? Ignore? Isolate?
806
isolate
redirect
ignore
808
Modify key name according to rule
810
Open key and return result
Use virtual key name for literal key name
812
Identify user-scoped candidate key name
814
816
Category of existence of the user-scoped candidate key ?
negative
positive
neutral
Key not found error
822
Identify application-scoped key as candidate key name
824
826
Category of existence of the application-scoped candidate key ?
negative
positive
neutral
818
Identify as literal key
Modify?
828
no
yes
838
Return permissions error
836
no
Permission?
yes
840
Copy to user isolation scope. Create placeholder keys (empty) in hierarchy as needed
842
Identify literal key
820
Open key and return result
830
Identify system scoped key name as candidate key name
832
Exist?
yes
no
834
Key not found error

Fig. 8

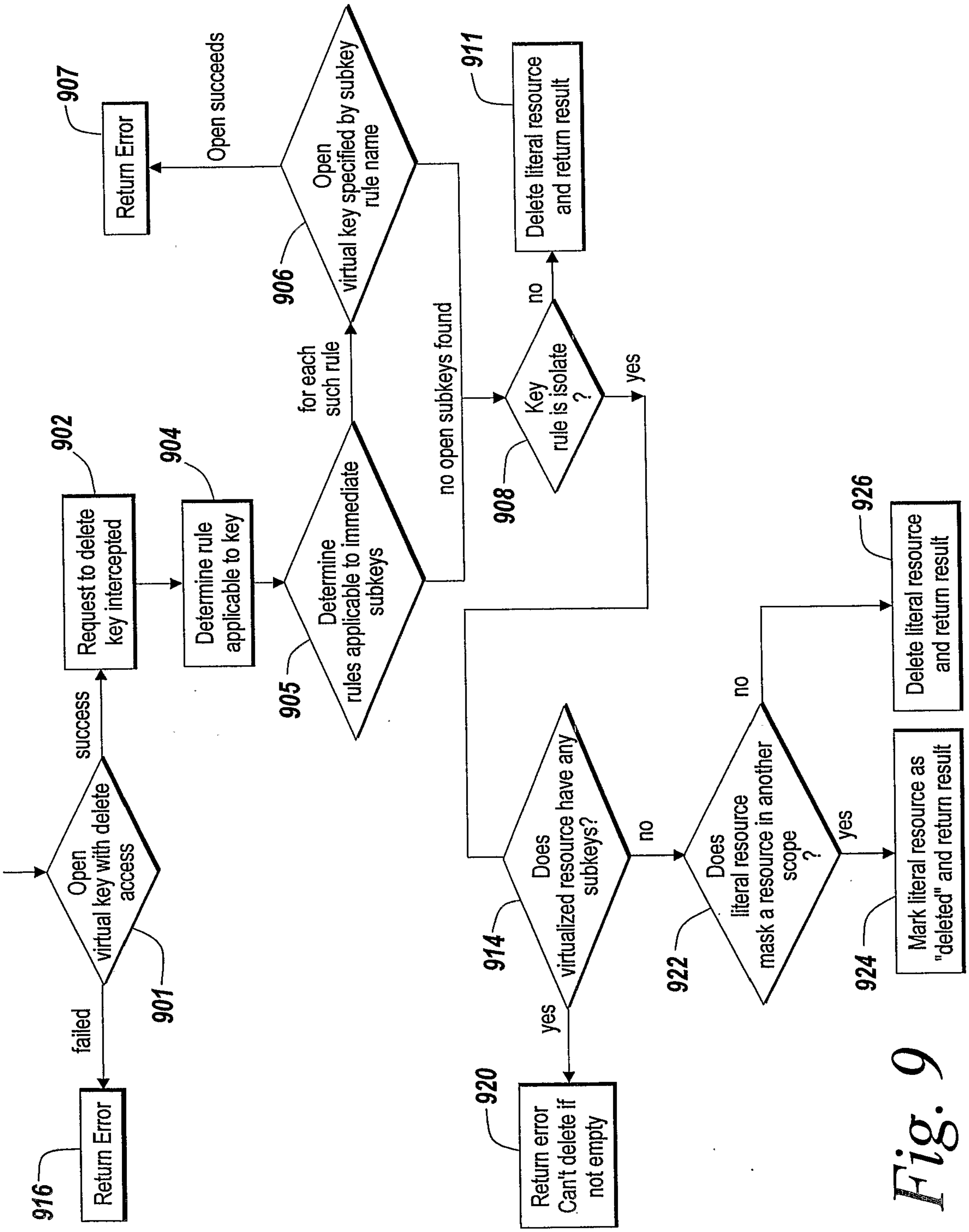

Open virtual key with delete access
901
failed
Return Error
916
success
Request to delete key intercepted
902
Determine rule applicable to key
904
Determine rules applicable to immediate subkeys
905
for each such rule
Open virtual key specified by subkey rule name
906
Open succeeds
Return Error
907
no open subkeys found
Key rule is isolate ?
908
no
Delete literal resource and return result
911
yes
Does virtualized resource have any subkeys?
914
yes
Return error Can't delete if not empty
920
no
Does literal resource mask a resource in another scope ?
922
no
Delete literal resource and return result
926
yes
Mark literal resource as "deleted" and return result
924

Fig. 9

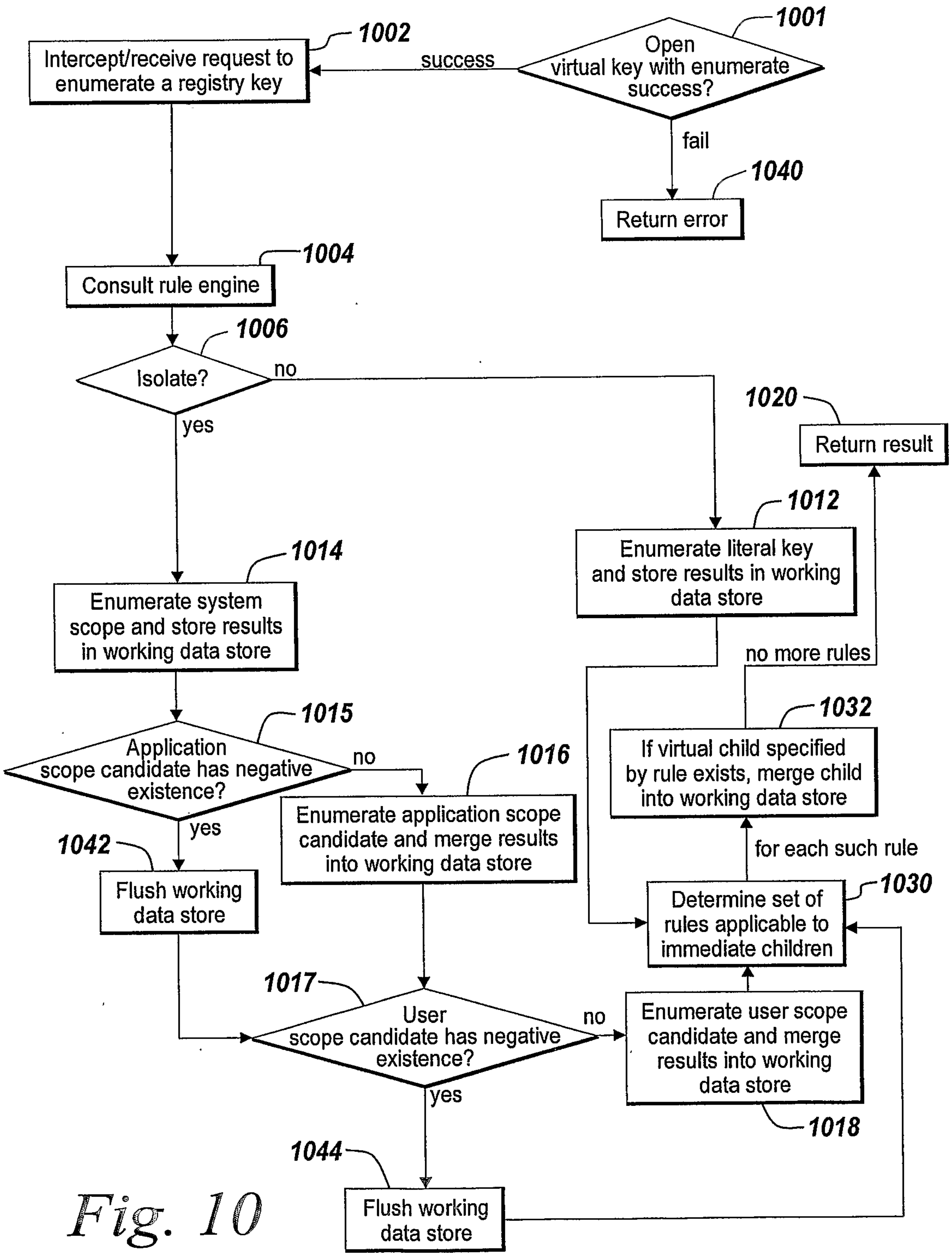

Open virtual key with enumerate success?
1001
success
fail
Intercept/receive request to enumerate a registry key
1002
Return error
1040
Consult rule engine
1004
Isolate?
1006
no
yes
Return result
1020
Enumerate literal key and store results in working data store
1012
Enumerate system scope and store results in working data store
1014
no more rules
If virtual child specified by rule exists, merge child into working data store
1032
Application scope candidate has negative existence?
1015
no
yes
Enumerate application scope candidate and merge results into working data store
1016
for each such rule
Flush working data store
1042
Determine set of rules applicable to immediate children
1030
User scope candidate has negative existence?
1017
no
yes
Enumerate user scope candidate and merge results into working data store
1018
Flush working data store
1044

Fig. 10

18/24

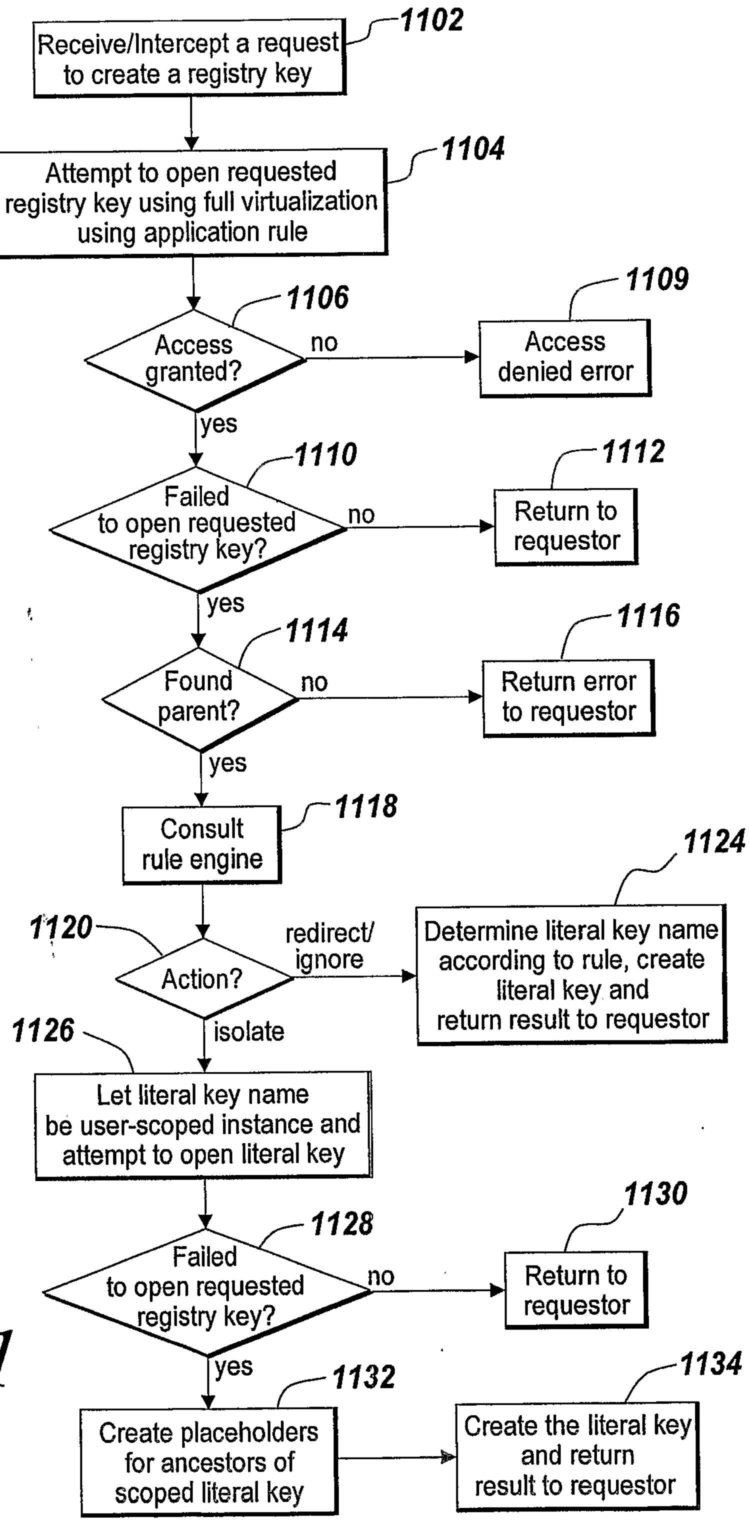

Receive/Intercept a request to create a registry key
1102
Attempt to open requested registry key using full virtualization using application rule
1104
Access granted?
1106
no
Access denied error
1109
yes
Failed to open requested registry key?
1110
no
Return to requestor
1112
yes
Found parent?
1114
no
Return error to requestor
1116
yes
Consult rule engine
1118
1120
Action?
redirect/ ignore
Determine literal key name according to rule, create literal key and return result to requestor
1124
isolate
1126
Let literal key name be user-scoped instance and attempt to open literal key
Failed to open requested registry key?
1128
no
Return to requestor
1130
yes
Create placeholders for ancestors of scoped literal key
1132
Create the literal key and return result to requestor
1134

Fig. 11

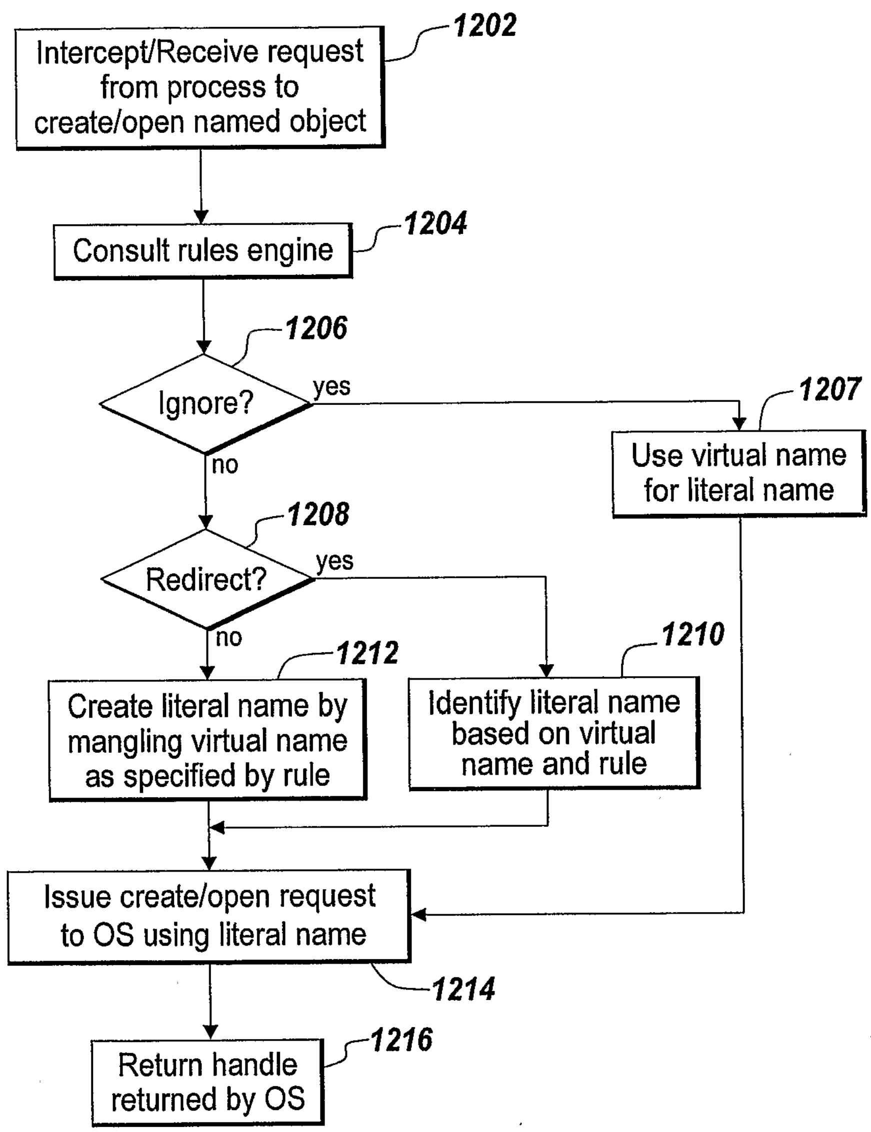

Intercept/Receive request from process to create/open named object
1202
Consult rules engine
1204
1206
Ignore?
yes
no
1207
Use virtual name for literal name
1208
Redirect?
yes
no
1212
Create literal name by mangling virtual name as specified by rule
1210
Identify literal name based on virtual name and rule
Issue create/open request to OS using literal name
1214
Return handle returned by OS
1216

Fig. 12

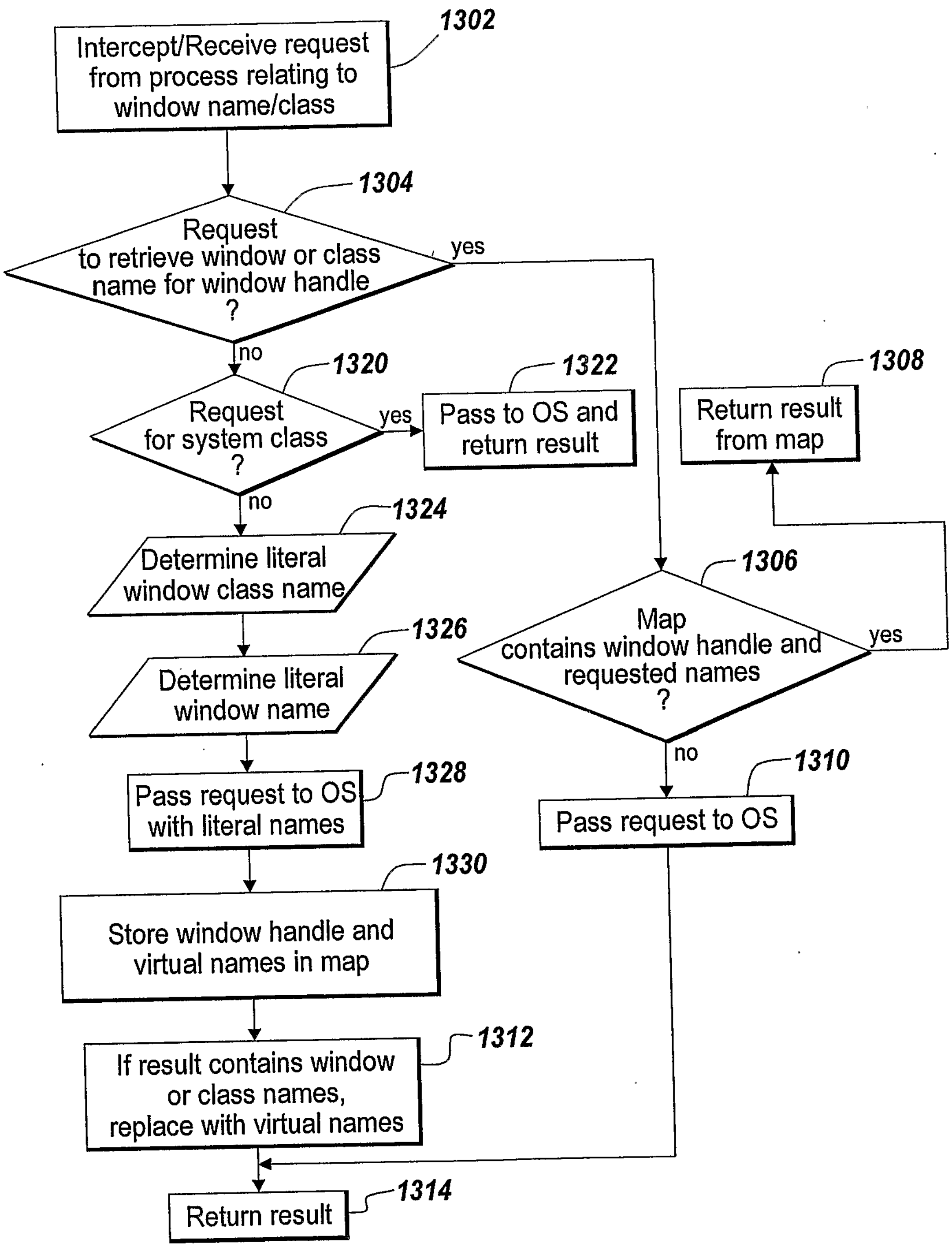

Intercept/Receive request from process relating to window name/class
1302
Request to retrieve window or class name for window handle ?
1304
yes
no
Request for system class ?
1320
yes
Pass to OS and return result
1322
no
Determine literal window class name
1324
Determine literal window name
1326
Pass request to OS with literal names
1328
Store window handle and virtual names in map
1330
If result contains window or class names, replace with virtual names
1312
Return result
1314
Return result from map
1308
Map contains window handle and requested names ?
1306
yes
no
Pass request to OS
1310

Fig. 13

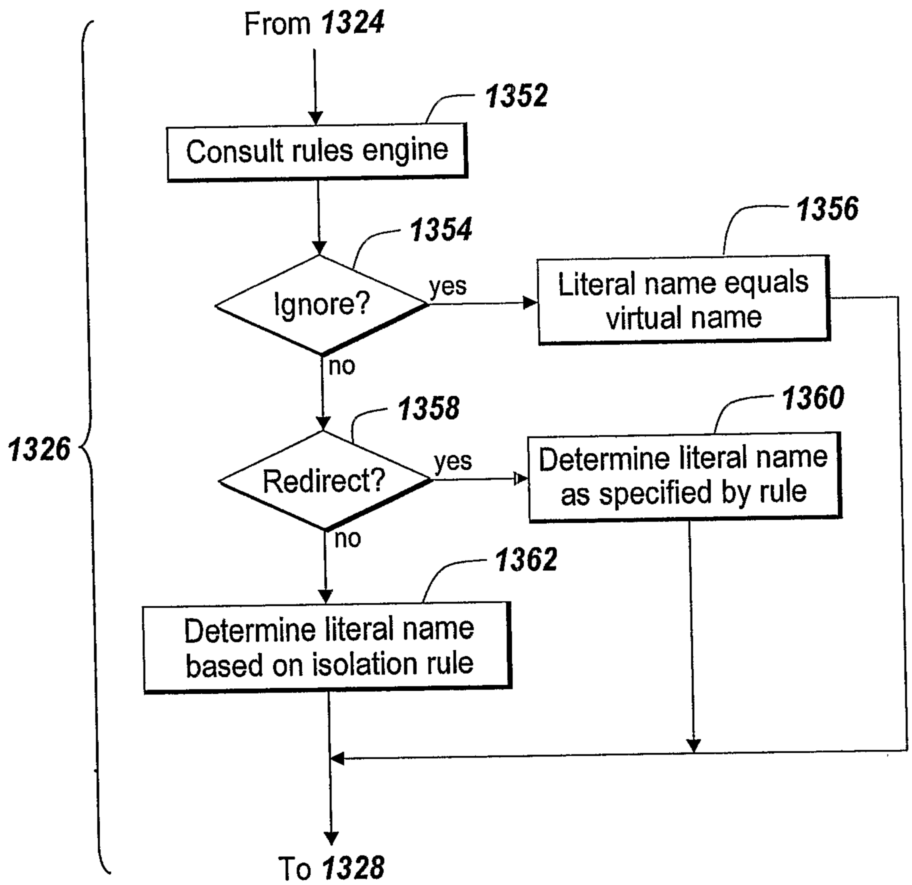

From 1324
1352
Consult rules engine
1354
Ignore?
yes
no
1356
Literal name equals virtual name
1358
Redirect?
yes
no
1360
Determine literal name as specified by rule
1362
Determine literal name based on isolation rule
1326
To 1328

Fig. 13A

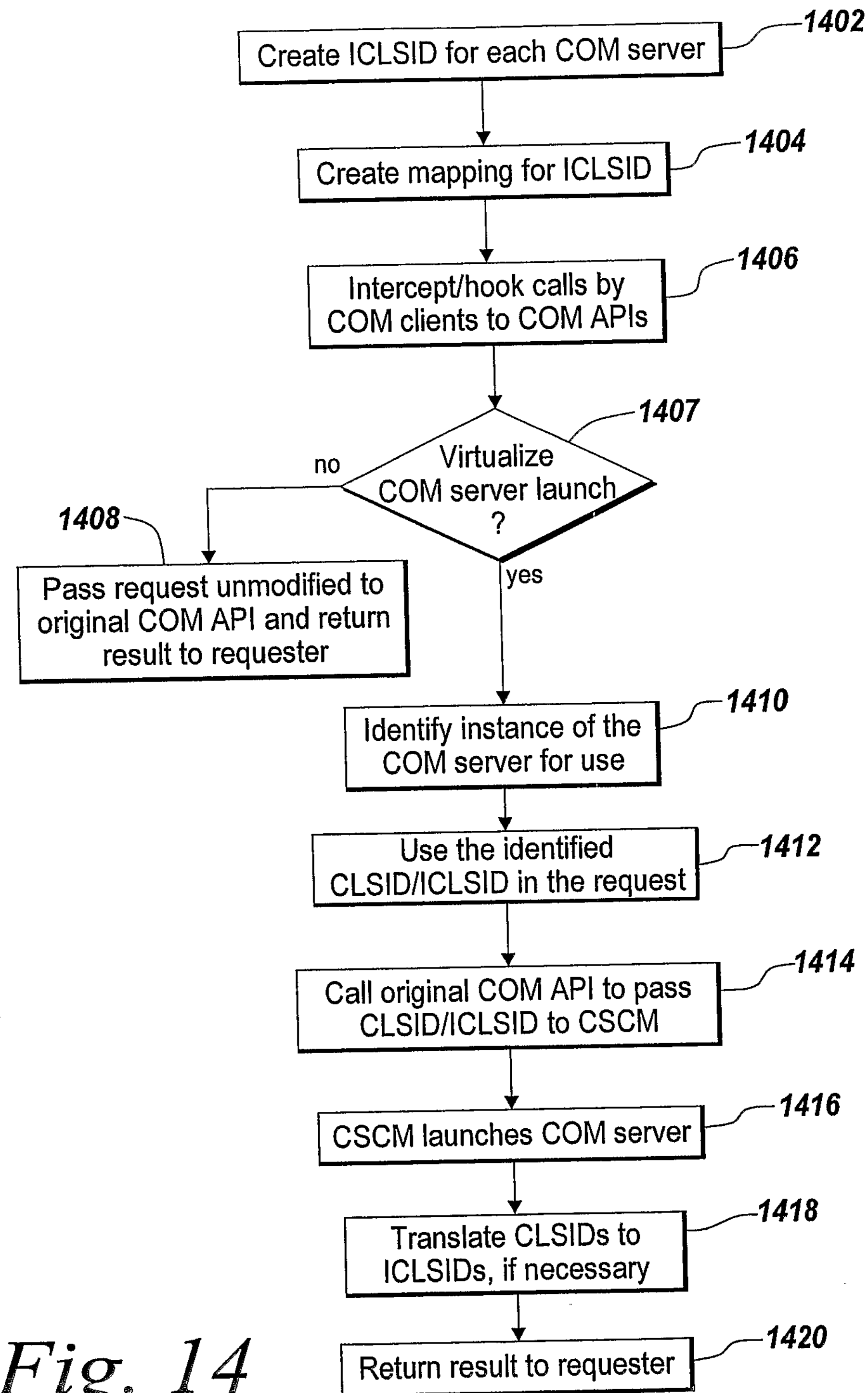

Create ICLSID for each COM server
1402
Create mapping for ICLSID
1404
Intercept/hook calls by COM clients to COM APIs
1406
Virtualize COM server launch ?
1407
no
yes
1408
Pass request unmodified to original COM API and return result to requester
Identify instance of the COM server for use
1410
Use the identified CLSID/ICLSID in the request
1412
Call original COM API to pass CLSID/ICLSID to CSCM
1414
CSCM launches COM server
1416
Translate CLSIDs to ICLSIDs, if necessary
1418
Return result to requester
1420

Fig. 14

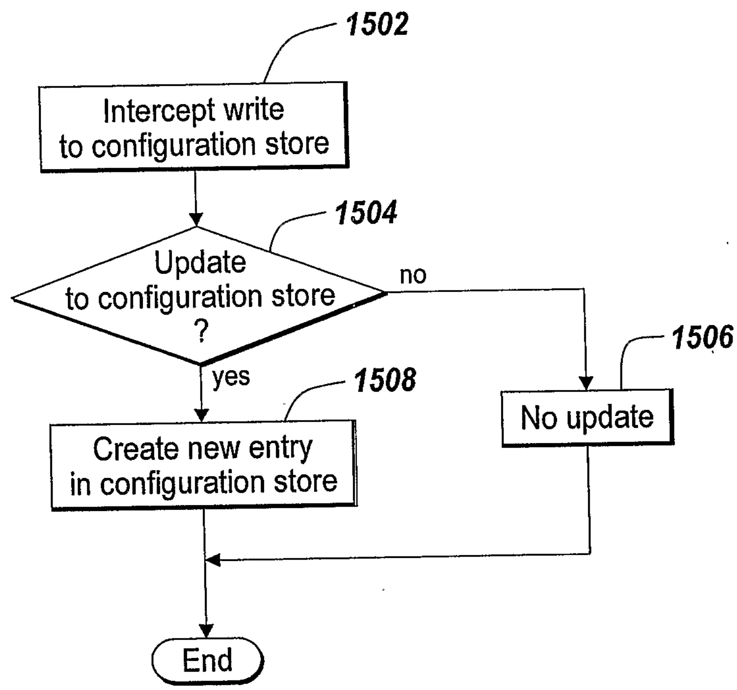

1502
Intercept write to configuration store
1504
Update to configuration store ?
no
yes
1508
1506
Create new entry in configuration store
No update
End

Fig. 15

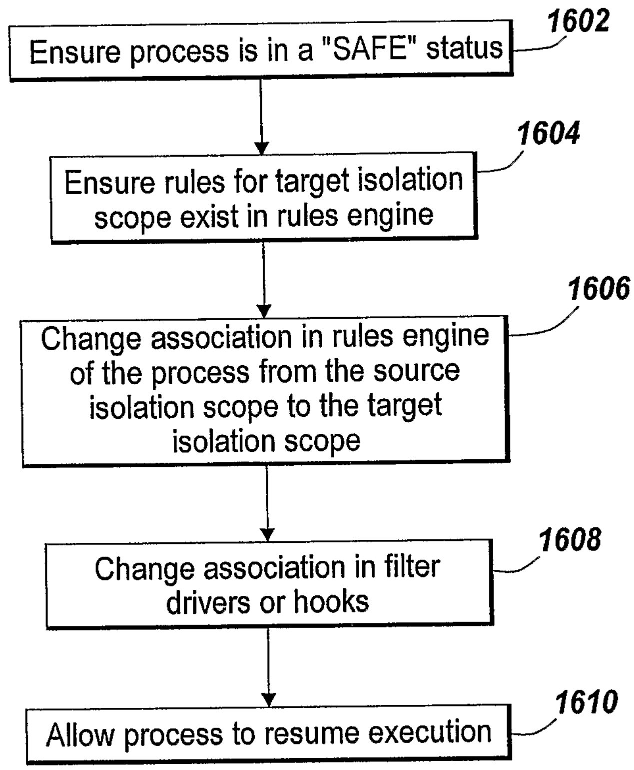

Ensure process is in a "SAFE" status
1602
Ensure rules for target isolation scope exist in rules engine
1604
Change association in rules engine of the process from the source isolation scope to the target isolation scope
1606
Change association in filter drivers or hooks
1608
Allow process to resume execution
1610

Fig. 16

INTERNATIONAL SEARCH REPORT

:ional application No
'US2005/033994

A. CLASSIFICATION OF SUBJECT MATTER
G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data, PAJ, IBM-TDB, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 01/25894 A (EJASENT INC; HIPP, BURTON, A; BHARADHWAJ, RAJEEV) 12 April 2001 (2001-04-12) page 1, line 12 - page 3, line 10 page 9, line 16 - page 18, line 30	1-26
X	US 2003/101292 A1 (FISHER JOSEPH A ET AL) 29 May 2003 (2003-05-29) page 1, left-hand column, paragraph 1 - page 1, right-hand column, paragraph 10	1-26
A	WO 01/95094 A (SUN MICROSYSTEMS, INC) 13 December 2001 (2001-12-13) page 1, line 1 - page 7, line 3	1-26
A	WO 00/45262 A (SUN MICROSYSTEMS, INC) 3 August 2000 (2000-08-03) page 2, line 4 - page 6, line 11	1-26

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
7 February 2006	15/02/2006

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Lo Turco, S

INTERNATIONAL SEARCH REPORT

al application No
...2005/033994

Patent document cited in search report		Publication date	Patent family member(s)			Publication date
WO 0125894	A	12-04-2001	AU	1074801	A	10-05-2001
			AU	1075101	A	10-05-2001
			AU	7864700	A	10-05-2001
			AU	7867000	A	10-05-2001
			AU	7867100	A	10-05-2001
			AU	7872100	A	10-05-2001
			AU	7996200	A	10-05-2001
			AU	8000800	A	10-05-2001
			WO	0125949	A1	12-04-2001
			WO	0126267	A1	12-04-2001
			WO	0125950	A1	12-04-2001
			WO	0125951	A1	12-04-2001
			WO	0126031	A2	12-04-2001
			WO	0125920	A1	12-04-2001
			WO	0125926	A1	12-04-2001
US 2003101292	A1	29-05-2003	NONE			
WO 0195094	A	13-12-2001	AU	6491401	A	17-12-2001
			EP	1299800	A2	09-04-2003
			US	6934755	B1	23-08-2005
WO 0045262	A	03-08-2000	AT	269558	T	15-07-2004
			AU	763958	B2	07-08-2003
			AU	4165700	A	18-08-2000
			CN	1338071	A	27-02-2002
			DE	1190316	T1	06-03-2003
			DE	60011615	D1	22-07-2004
			DE	60011615	T2	07-07-2005
			EP	1190316	A2	27-03-2002
			JP	2003518279	T	03-06-2003
			US	6907608	B1	14-06-2005
			US	2005102679	A1	12-05-2005

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 January 2002 (24.01.2002)

PCT

(10) International Publication Number
WO 02/06941 A2

(51) International Patent Classification[7]: G06F 3/00

(21) International Application Number: PCT/US01/22278

(22) International Filing Date: 16 July 2001 (16.07.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/617,669 17 July 2000 (17.07.2000) US

(71) Applicant: CONNECTIX CORPORATION [US/US]; 2955 Campus Drive, Suite 100, San Mateo, CA 94403 (US).

(72) Inventors: TRAUT, Eric, P.; 3 Iris Lane, San Carlos, CA 94070 (US). MARTZ, Benjamin; 258 W. 40th Ave., San Mateo, CA 94403 (US).

(74) Agents: FULGHUM, Roger; Baker Botts L.L.P., One Shell Plaza, 910 Louisiana, Houston, TX 77002 et al. (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

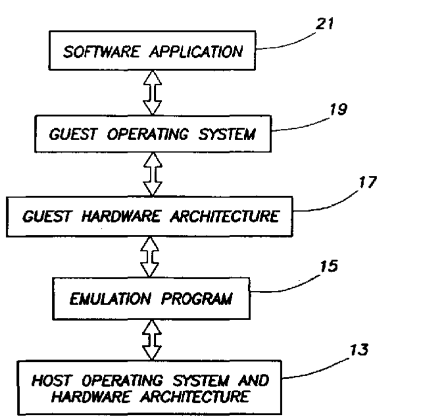
(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

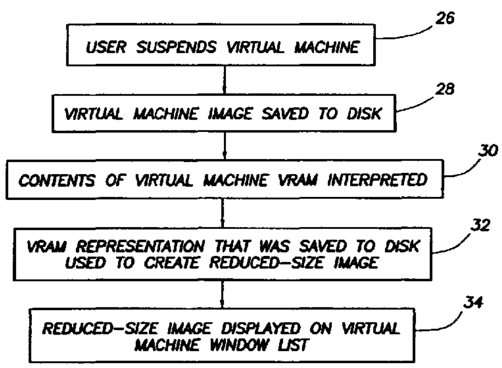
Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR DISPLAYING CURRENT IMAGES OF VIRTUAL MACHINE ENVIRONMENTS

(57) Abstract: A system and method for displaying thumbnail images of the video output of one or more software applications in a window or similar graphical interface to allow the user of a computer system to conveniently and quickly monitor the overall status and progress of several software applications that are running simultaneously. The thumbnail images are generated from the VRAM associated with the software application and are preferably displayed with information corresponding to the associated software application. The thumbnail images may be static or generated at regular intervals according to user preference and the status of the software application.





WO 02/06941 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEM AND METHOD FOR DISPLAYING CURRENT IMAGES OF VIRTUAL MACHINE ENVIRONMENTS

TECHNICAL FIELD OF THE INVENTION

The present disclosure relates in general to the field of virtual machines, and, more particularly, to a system and method for displaying images from virtual machine environments.

BACKGROUND OF THE INVENTION

Computers include general purpose central processing units (CPUs) that are designed to execute a specific set of system instructions. A group of processors that have similar architecture or design specifications may be considered to be members of the same processor family. Examples of current processor families include the Motorola 680X0 processor family, manufactured by Motorola, Inc. of Phoenix, Arizona; the Intel 80X86 processor family, manufactured by Intel Corporation of Sunnyvale, California; and the PowerPC processor family, which is manufactured by Motorola, Inc. and used in computers manufactured by Apple Computer, Inc. of Cupertino, California. Although a group of processors may be in the same family because of their similar architecture and design considerations, processors may vary widely within a family according to their clock speed and other performance parameters.

Each family of microprocessors executes instructions that are unique to the processor family. The collective set of instructions that a processor or family of processors can execute is known as the processor's instruction set. As an example, the instruction set used by the Intel 80X86 processor family is incompatible with the instruction set used by the PowerPC processor family. The Intel 80X86 instruction set is based on the Complex Instruction Set Computer (CISC) format. The Motorola PowerPC instruction set is based on the Reduced Instruction Set Computer (RISC) format. CISC processors use a large number of instructions, some of which can perform rather complicated functions, but which require generally many clock cycles to execute. RISC processors use a smaller number of available instructions to perform a simpler set of functions that are executed at a much higher rate.

The uniqueness of the processor family among computer systems also typically results in incompatibility among the other elements of hardware architecture of the computer systems. A computer system manufactured with a processor from the Intel 80X86 processor family will have a hardware architecture that is different from the hardware architecture of a computer system manufactured with a processor from the PowerPC processor family. Because of the uniqueness of the processor instruction set and a computer system's hardware architecture, application software programs are typically written to run on a particular computer system running a particular operating system.

A computer manufacturer will want to maximize its market share by having more rather than fewer applications run on the microprocessor family associated with the computer manufacturer's product line. To expand the number of operating systems and application programs that can run on a computer system, a field of technology has developed in which a given computer having one type of CPU, called a host, will include an emulator program that allows the host computer to emulate the instructions of an unrelated type of CPU, called a guest. Thus, the host computer will execute an application that will cause one or more host instructions to be called in response to a given guest instruction. Thus, the host computer can both run software design for its own hardware architecture and software written for computers having an unrelated hardware architecture. As a more specific example, a computer system manufactured by Apple Computer, for example, may run operating systems and program written for PC-based computer systems. It may also be possible to use an emulator program to operate concurrently on a single CPU multiple incompatible operating systems. In this arrangement, although each operating system is incompatible with the other, an emulator program can host one of the two operating systems, allowing the otherwise incompatible operating systems to run concurrently on the same computer system.

When a guest computer system is emulated on a host computer system, the guest computer system is said to be a virtual machine, as the host computer system exists only as a software representation of the operation of the hardware architecture of the guest computer system. The terms emulator and virtual machine are sometimes used interchangeably to denote the ability to mimic or emulate the hardware architecture of an entire computer system. As an example, the Virtual PC software

created by Connectix Corporation of San Mateo, California emulates an entire computer that includes an Intel 80X86 Pentium processor and various motherboard components and cards. The operation of these components is emulated in the virtual machine that is being run on the host machine. An emulator program executing on the operating system software and hardware architecture of the host computer, such as a computer system having a PowerPC processor, mimics the operation of the entire guest computer system. The emulator program acts as the interchange between the hardware architecture of the host machine and the instructions transmitted by the software running within the emulated environment.

Over the years, the number of operating systems able to execute on a given processor family has increased markedly. For example, at present, the Windows operating system alone has several versions, such as Windows 3.1, Windows 95, Windows 98, Windows 98SE, Windows 2000, Windows NT, and Windows Millennium. Thus, a user of a computer system having an Intel 80X86 processor architecture may choose among several operating systems. Similarly, if the hardware architecture of a computer system running an Intel 80X86 architecture is being emulated in a host computer system, it may be desirable to run several virtual machines simultaneously, with each virtual machine operating according to a different operating system. As an example, it may be desirable to emulate on a Macintosh computer system several guest computers systems, each running a separate version of Windows. By doing so, the Macintosh user can take advantage of the wide variety of software applications designed to run on PC computer systems.

As the user increases the number of software applications and operating systems that is running on a computer system, it becomes more difficult for the user to manage the various tasks performed on the computer system. Accordingly, as the user increases the number of virtual machines running on the computer system, it becomes more difficult for the user to keep track of both the virtual machines and the respective applications running on the computer system's native operating system and the various virtual machines. For example, in operating systems that use windowing environments, such as Windows, the Macintosh operating system, and OS/2, for example, users may resize or move windows that correspond to various applications. Windows that are minimized or suspended are typically moved to a task bar, application bar or

application list where they can be later accessed. Unfortunately, these application bars or lists generally do not present the user with enough information about the particular applications contained therein to allow the user to quickly find the particular application the user is interested in resuming. This problem is exacerbated as the number of applications and corresponding windows increases.

SUMMARY OF THE INVENTION

In accordance with teachings of the present disclosure, a system and method for managing multiple virtual computer environments are disclosed that provide significant advantages over prior developed systems.

The system and method described herein allow a user of a host computer system running multiple virtual machine environments to view the thumbnail images of the video output of each virtual machine application. According to one embodiment of the present invention, the thumbnail images are generated by interpreting the contents of the virtual machine's video RAM (VRAM) and scaling the resulting representations into several thumbnail images that may be conveniently viewed, for example, in a single window. Interpretation of the virtual machine's VRAM is generally dependent on the current video mode of the virtual machine, such as text, planar video, linear video, among other examples. According to one embodiment of the present invention, the thumbnail images are generated using bilinear sampling techniques to create an anti-aliased miniature image. In the case of active emulated computer systems, thumbnail images are preferably continuously generated to give a real-time representation of the virtual machine application within the thumbnail view. The thumbnail or reduced-size images are preferably saved whenever the respective virtual machine is placed in a suspension mode and accordingly saved to the storage device of the computer system running the virtual machine software. The image of these suspended images are likewise displayed for the user, allowing the user to quickly view the status of the virtual machine regardless of whether the virtual machine is active or suspended.

The disclosed system and method provide several technical advantages over conventional systems and methods for managing multiple computer applications, such as multiple virtual computer environments. One advantage of the system and method

of the present invention is that it allows a user of a computer system that is running multiple virtual machine environments to view the status of several multiple virtual machine environments at one glance. This allows the user to easily manage several virtual machine environments that are running simultaneously. Another advantage of the present system and method is that the user may quickly ascertain the status of a given virtual machine without having to activate or unsuspend the virtual machine. This allows the user to save time, especially if the virtual machine is running at a relatively slow speed, or if the user is working on several projects.

Other technical advantages should be apparent to one of ordinary skill in the art in view of the specification, claims, and drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the present embodiments and advantages thereof may be acquired by referring to the following description taken in conjunction with the accompanying drawings, in which like reference numbers indicate like features, and wherein:

Figure 1 is a diagram of the logical relationship of the elements of an emulated computer system running in a host computer system;

Figure 2 is diagram depicting the screen display from a computer system utilizing an embodiment of the present invention;

Figure 3 is a flow diagram of method for displaying a thumbnail image of an emulated computer system; and

Figure 4 is a flow diagram of the manner in which the emulator program polls each virtual machine environment to determine the status of the virtual machine and to locate the reduced-size image to be displayed for each virtual machine

DETAILED DESCRIPTION OF THE INVENTION

Shown in Figure 1 is a diagram of the logical layers of the hardware and software architecture for an emulated operating environment in a computer system 11. An emulation program 15 runs on a host operating system and/or hardware architecture 13. Emulation program 15 emulates a guest hardware architecture 16 and a guest operating system 19. Software application 21 in turn runs on guest operating system

19. In the emulated operating environment of Figure 1, because of the operation of emulation program 15, software application 21 can run on the computer system 11 even though software application 21 is designed to run on an operating system that is generally incompatible with the host operating system and hardware architecture 13.

Shown in Figure 2 is a screen display 10 from a computer system 12 that implements the system and method of the present invention. Computer system 12 may use any type of processor and any type of operating system that are suitable for running one or more virtual machines. For example, computer system 12 may include a processor from one of several processor families such as the PowerPC processor, the Intel 80X86 Pentium processor, or the Motorola 680X0 processor, among other examples. The operating system of computer system 12 is preferably an operating system that may use a windowing environment. Examples of suitable operating systems include Windows, the Macintosh OS, or OS/2, among other examples. For example, the computer system 12 depicted in Figure 2 uses the Macintosh OS and a PowerPC processor.

Computer system 12 is running one or more virtual machines or emulators on its native or host operating system. As discussed above, an emulator or virtual machine allows a host computer to run a software application that is designed for an operating system or processor that is not native to the host computer. Thus, computer system 12 is a host computer system. If the native operating system for computer system 12 uses a windowing environment or other similar graphical user interface that represents programs, files, and options by means of icons, menus, and dialog boxes on the screen, then the virtual machines or emulators may be displayed in a virtual machine list window 14 that can be displayed on the desktop 16 of the operating system of computer system 12. From the virtual machine list window 14, the user of the computer system may monitor or modify the function of the virtual machines or emulators running on computer system 12. The user may modify the function of the virtual machines or emulators using control buttons 18 or any similar graphical interface functionality. For example, the user may choose additional virtual machines or emulators to run, choose new software applications to run on virtual machines or emulators that are already running, modify the settings associated with any of the virtual machines or emulators,

terminate any of the virtual machines or emulators, or suspend any of the virtual machines or emulators, among other examples.

One or more virtual machine status windows are displayed within virtual machine list window 14. Shown in list window 14 are virtual machine status windows 20a-20i. Each virtual machine status window is preferably associated with one of the virtual machines running on computer system 12. Alternatively, if a particular virtual machine is running more than one software application, then a virtual machine status window can be associated with each software application. Each virtual machine status window displays information regarding the associated virtual machine or emulator. For example, the virtual machine status window may display information regarding the operating system or computer system that is being emulated, the software application that is running under the associated virtual machine or emulator, and the current status of the software application, among other examples.

In addition, each virtual machine status window also contains a thumbnail image. Shown in list window are thumbnail images 22a-22i, each of which is associated with a virtual machine status window 20a-20i. Each thumbnail image is a representation of the image or graphical output that is currently being generated by the software application or operating system running under the virtual machine associated with the respective virtual machine status window. If the software application is currently suspended or saved, then the associated thumbnail image is the image that was saved in VRAM of the virtual machine at the time that the virtual machine environment was suspended. Each thumbnail image is preferably sized to allow the user to conveniently view all or several of the virtual machine status windows 20 on the computer screen of computer system 12. If the operating system of the virtual machines displays images in color, then thumbnail image is preferably also displayed in color. As discussed below, thumbnail image may be a static image or an image that is updated in real-time or periodically.

The computer system 12 illustrated in Figure 2 is using the Macintosh OS and is running through an emulator program several virtual PC machines on the host Macintosh OS. The virtual machine status windows 20a-20i in Figure 2 display information associated with the various virtual PC machines that are operating on computer system 12. Virtual machine status window 20a indicates that computer

system 12 includes a virtual machine that is emulating a computer system with a DOS operating system. Virtual machine status window 20a also indicates that the operation of a video game software application is associated with this virtual machine. In addition, virtual machine status window 20a indicates that the operation of this video game is currently saved or suspended. Because the software application is suspended, thumbnail image 22a is a representation of the image that was in VRAM of the virtual machine at the time that the virtual machine was suspended. Similarly, virtual machine status windows 20b through 20i indicate that computer system 12 also includes virtual machines that emulate computer systems running Windows 98, Windows 2000, Windows 3.1, Windows 95, Windows 98SE, Windows Millennium, and Windows NT operating systems. Thumbnail images 20b through 20i are representations of the images that were created by the operating systems or software applications associated with virtual machine status windows 20b through 20i. The term Windows is a trademark of Microsoft Corporation.

Generally, each software application running on an operating system implementing a windowing environment will be running in its own window. Therefore, the other active software applications of the computer system 12 will be running in one of windows 24. Preferably, the user does not operate the software application depicted in the virtual machine list window 14 directly from the virtual machine list window 14. Thus, if software application depicted in virtual machine list 14 is running, it will be running on an active window 24 in addition to being depicted in virtual machine window list 14. However, the user may preferably be able to restore or maximize the window 24 associated with a software application running under a virtual machine or emulator application by clicking on, or otherwise activating, the virtual machine status window 22 associated with that software application and the corresponding virtual machine or emulator.

Figure 3 depicts a process diagram for suspending the contents of a virtual machine and later displaying those contents in a reduced-size representation for the user. As discussed above, the user of computer system 12 may have several virtual machines or emulated operating system environments running at the same time. Due to resource limitations, a computer system 12 running several virtual machine applications simultaneously may incur adverse performance effects. Therefore, it may be desirable

for a user of computer 12 to suspend a particular virtual machine or operating system environment. At step 26 of Figure 3, the user of computer system 12 suspends a virtual machine application. By temporarily halting the virtual machine, the user can conserve resources and improve the management or performance of other applications. When a virtual machine application is suspended, the operating parameters associated with the virtual machine are stored, so that the operation of the virtual machine can be quickly resumed if the user of computer system 12 decides to activate or unsuspend the virtual machine. Thus at step 28, the contents of the RAM, VRAM, registers, parameters and other data associated with the virtual machine application are saved to a storage device of computer system 12. Typically, the storage device will be a hard drive, disk, or some other nonvolatile storage medium. For instance, if the virtual machine is emulating a processor, network card and video card of a particular hardware architecture, then the registers for these various components must be saved to memory as part of the suspension process. Saving the contents of VRAM of the virtual machine saves the video image most recently displayed by the virtual machine. Similarly, various parameters for these components must be saved, such as the video mode and resolution for the video card, for example. The step of saving the virtual machine environment to a storage device is preferably performed with a compression algorithm or similar functionality to reduce the storage space or time required for this step.

To successfully manage a computer system 12 with several virtual machines, the user must be able to quickly ascertain the status of each virtual machine, such as whether the virtual machine is running a software application, and whether the virtual machine is active or suspended. In the case of suspended virtual machines, after the suspended virtual machine environment has been saved to the storage device, the thumbnail image 22 and other information must be generated for the virtual machine status window 20. To generate thumbnail image 22, the contents of the virtual machine's VRAM, which have been stored to memory, must be interpreted at step 30. Interpreting the virtual machine's VRAM involves taking into account the video mode or video adapter setting of the virtual machine at the time it was suspended. The video mode is the manner in which the virtual machine's video adapter displays on-screen images to the monitor of computer system 12. The most common video modes are text mode and graphics mode. In text mode, only characters such as letters, numbers, and

symbols may be displayed, and these characters are not drawn as pixel representations. On the other hand, graphics mode produces all monitor images, whether text or art, as patterns of pixels. Examples of graphics modes include planar video and linear video, among other examples. Another consideration for interpreting the VRAM is the associated bit depth or color depth. Because color in a computer image is a function of the number of bits available to provide different shades for each pixel, bit depth is a measure of the number of different colors that may be displayed in the image. Other graphical settings and parameters may need to be processed or taken in account as well. In the case of active virtual machines, the contents of the VRAM for active virtual machines is saved in the main memory of the host system. To display a reduced-size image of the active virtual machine, the contents of the main memory of the host system must be likewise interpreted.

With reference to Figure 3, once the contents of the virtual VRAM have been interpreted, the resulting representation is used to create thumbnail image 22 at step 32. At this point, the contents of the virtual VRAM have been saved to disk, and this saved image from disk is the image that is used to create the thumbnail image 22. Generally thumbnail images 22 are preferably created using bilinear sampling techniques to create an anti-aliased miniature image. Anti-aliasing, or oversampling, is a graphics technique that smoothes out the jagged appearance of curved or diagonal lines, a phenomenon known as "jaggies." Examples of anti-aliasing techniques include surrounding selected pixels in the image with intermediate shades and manipulating the size and alignment of the pixels. An anti-aliased image is preferred because the thumbnail image 22 is preferably small and the presence of jaggies or other visual distortions will typically result in a poor image. A bilinear sampling technique is used to downsample the source image on the VRAM by averaging the pixels in the source image to create a scaled-down thumbnail image 22. Another technique that may be used to create thumbnail image 22 is spot or point sampling to map a pixel in the source image to a pixel in the thumbnail image 22. Unfortunately, point sampling tends to result in a significant loss of data. Other techniques suitable for creating a scaled-down image may be used, such as area sampling and gaussian filtering, among other examples. The thumbnail image 22 is then displayed in the virtual machine list window 14 at step 34. Preferably, thumbnail image 22 is saved to the storage device along with the virtual

machine that is suspended so that the thumbnail image 22 may be quickly displayed at a later time, as the image does not need to be created again from the VRAM.

In the case of active virtual machines, a user of computer system 12 may wish to keep several virtual machine applications active and monitor their overall performance from time to time while directly operating the virtual machine application the user has chosen as a primary action item. For example, the user of computer system 12 may be running a virtual machine that is running a word processor software application for the purposes of creating a document and at least one other virtual machine that is running computer-aided design software applications (CAD). While the process of creating a document requires continuous user input to the word processor software application, the CAD applications do not necessarily require continuous user input if, for example, the CAD applications are rendering wire-frame skeleton models. However, the user may be interested in monitoring the progress of the CAD applications from time to time, but preferably only at a cursory level, as the user is only interested in gaining an overall impression of the rendering job before returning his attention to the word processor application. Therefore, the technique of the present invention allows the user, in the case of active virtual machines, to monitor a thumbnail image of the active virtual machines on a real time basis.

Shown in Figure 4 is a flow diagram of the manner in which the emulator program polls each virtual machine environment to determine the status of the virtual machine and to locate the reduced-size image to be displayed for each virtual machine. At step 36, the emulator routine scans through each emulated system. For each system, the emulator program performs the decision of step 38 and then steps through the remainder of the flow diagram of Figure 4. If the virtual machine is determined to be active at step 38, then at step 40 a reduced-size image of the VRAM of the virtual machine is interpreted by the emulator program and displayed for the user. In the case of active emulated systems, the contents of their VRAM are maintained by the host system in the main memory of the host system. If it is determined at step 38, that the virtual machine is not active, processing continues at step 42, where it is determined whether the virtual machine has been suspended, *i.e.*, whether the image contents of the virtual machine have been saved to disk. If the image contents of the virtual machine have been suspended to disk, the emulator program at step 44 loads from disk and

displays a thumbnail image of the image of the virtual machine at the time of its suspension. If it is determined at step 42 that the virtual machine is not suspended, then at step 46 a blank reduced-size image is loaded and displayed for the user. After each of steps 40, 42, or 46, the processing continues at step 36 with a scan of each virtual machine. The processing steps of Figure 4 permit the emulator program to continually monitor the status of each virtual machine. If a virtual machine transitions from suspended mode to active mode, for example, the emulator program is able to monitor this transition and display the correct image of the virtual machine. Additionally, for those virtual machines that are active, the processing steps of Figure 4 permit the emulator to display for the user a continually updated rendering of the image of the active virtual machine. The processing steps of Figure 4 may occur roughly once per second. At this processing rate, a continually update image of each active machine is displayed for the user to permit the user to monitor the state of each active virtual machine.

Although the disclosed embodiments have been discussed in reference to the field of virtual machines and emulators, it can be readily appreciated that the present invention has equal applicability to other types of software applications. For example, the user of computer system 12 may be running several software applications simultaneously and would like to monitor the status of all of the applications in a convenient manner. The virtual machine list window 14 could alternatively display all of the software applications that are currently active or otherwise on the desktop 16. Each software application would be associated with a virtual machine status window 20 that would provide information on the software application and its status. A thumbnail image 22 would be generated for each virtual machine status window 20 that would represent the video or graphical output of the software application. As discussed above, the thumbnail image 22 could be static or continuously updated depending on the user's preference and the status of the associated software application.

It should be noted that the choice of computer system that can be emulated according to the technique of the present invention is not limited to computers systems that operate according to the Intel 80X86 system architecture and run a Windows operating system. The invention described herein applies to any emulated environment in which one or more virtual machines can be displayed in a windowed setting. Thus,

the invention disclosed herein can be used in any environment in which the contents of the virtual video memory buffer may be displayed for the user. The method disclosed herein is especially advantageous in that it permits a user to monitor the activities or suspended state of one or more virtual machine environments. Thus, the user, when operating in a first virtual machine environment can readily determine the operating state or suspended condition of one or more other virtual machine environments that typically comprise unique operating system environments. The reduced-size representations of the virtual machine environments are rendered on the basis of the contents of the saved virtual VRAM of each environment. Thus, from a location in main memory, the emulator program can create for the user an easily accessible visual directory of the other virtual machine environments being emulated by the host system.

Although the disclosed embodiments have been described in detail, it should be understood that various changes, substitutions, and alterations can be made to the embodiments without departing from their spirited scope.

WHAT IS CLAIMED IS:

1. A computer system for running one or more software applications, wherein the software applications are suitable for generating a video output, comprising:

a host operating system suitable for displaying a graphical user interface;

multiple emulated operating systems being emulated by one or more emulator programs running on the host operating system; and

wherein the host operating system is able to display for a user a reduced-size representation of the video output of the emulated operating systems that are being operated in a background mode.

2. The computer system of claim 1, further comprising one or more virtual video memory components suitable for storing the video output of the emulated operating systems.

3. The computer system of claim 2, wherein one or more of the video memory components are VRAM memory.

4. The computer system of claim 1,

wherein the emulated operating systems operating in a background mode are active; and

wherein the thumbnail images for the emulated operating systems are generated from the video information stored on the video memory components at predetermined intervals while the software applications are active.

5. The computer system of claim 4, wherein the predetermined intervals are such that the thumbnail images are real-time representations of the video output from the active software applications.

6. The computer system of claim 1,

wherein the graphical user interface is a windowing environment suitable for displaying one or more windows; and

wherein the portion of the graphical user interface comprising the reduced-size representation is a window.

7. The computer system of claim 1, wherein the reduced-size representations are created using a bilinear sampling technique.

8. A computer system for running one or more software applications, wherein the software applications are suitable for generating a video output, comprising:

a host operating system suitable for displaying a graphical user interface;

multiple emulated virtual machines being emulated by one or more emulator programs running on the host operating system; and

wherein the host operating system is able to display for a user a reduced-size representation of the video output of each virtual machine being operated in a background mode.

9. The computer system of claim 8, wherein the reduced-size representations are representations of the video outputs of the virtual machines that are being operated in the background mode.

10. The computer system of claim 9,

further comprising a virtual video memory associated with each of the virtual machines; and

wherein the reduced-size representations are generated from the video information stored in the virtual video memory associated with each virtual machine.

11. A method for displaying a reduced-size image of multiple emulated computer systems, comprising the steps of:

suspending one or more of the multiple emulated computer systems by saving to memory in the host computer system the image of the emulated computer system;

reading in at the emulator program from memory in the host computer system the image of the suspended emulated computer system;

interpreting in the emulator program the contents of the saved image of the suspended emulated computer system;

displaying a reduced-size representation of the suspended emulated computer system.

12. A method for displaying a reduced-size image of multiple emulated computer systems, comprising the steps of:

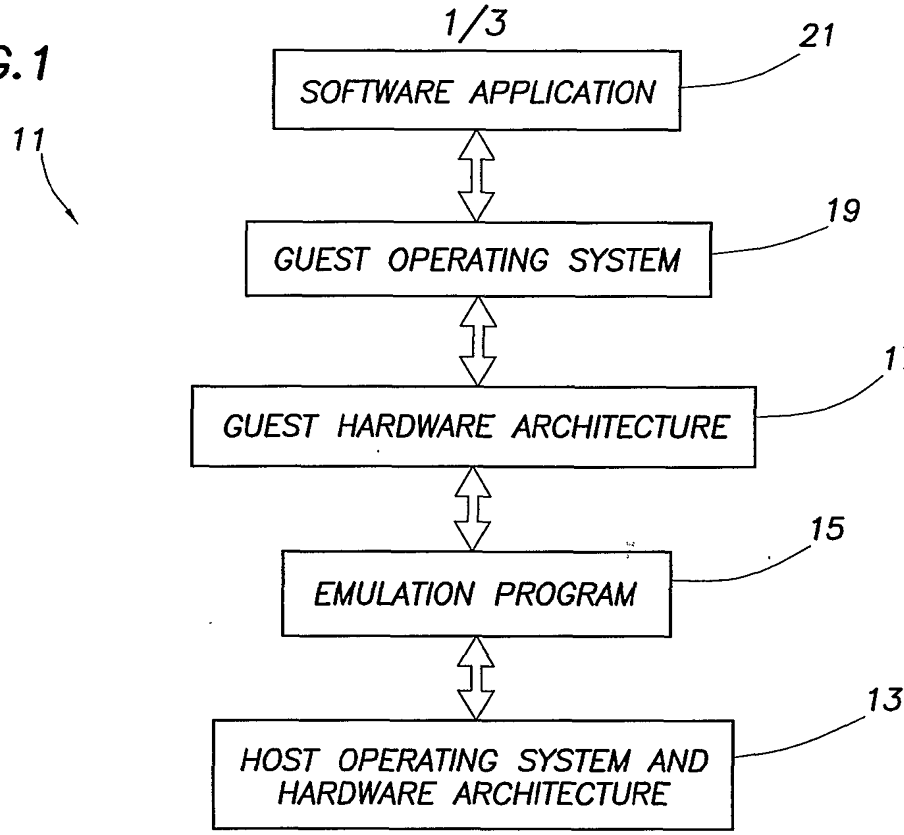
reading in at the emulator program from memory in the host computer system the image of the emulated computer system;

interpreting in the emulator program the contents of the image of the emulated computer system;

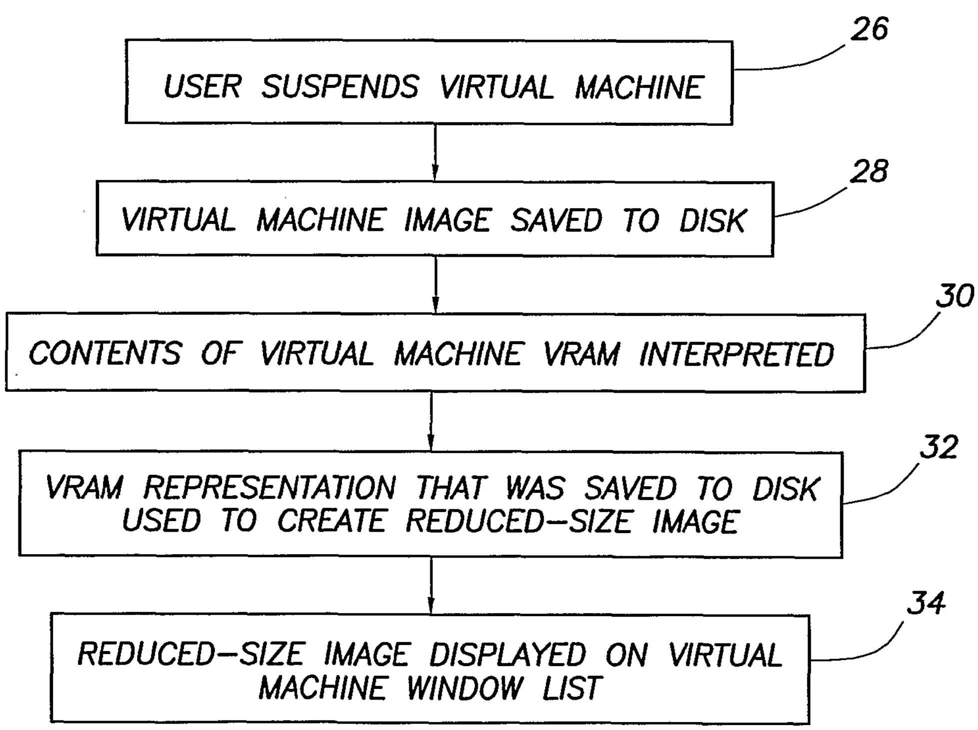
displaying a reduced-size representation of the emulated computer system;

periodically updating the reduced-size representation of the emulated computer system.

FIG. 1

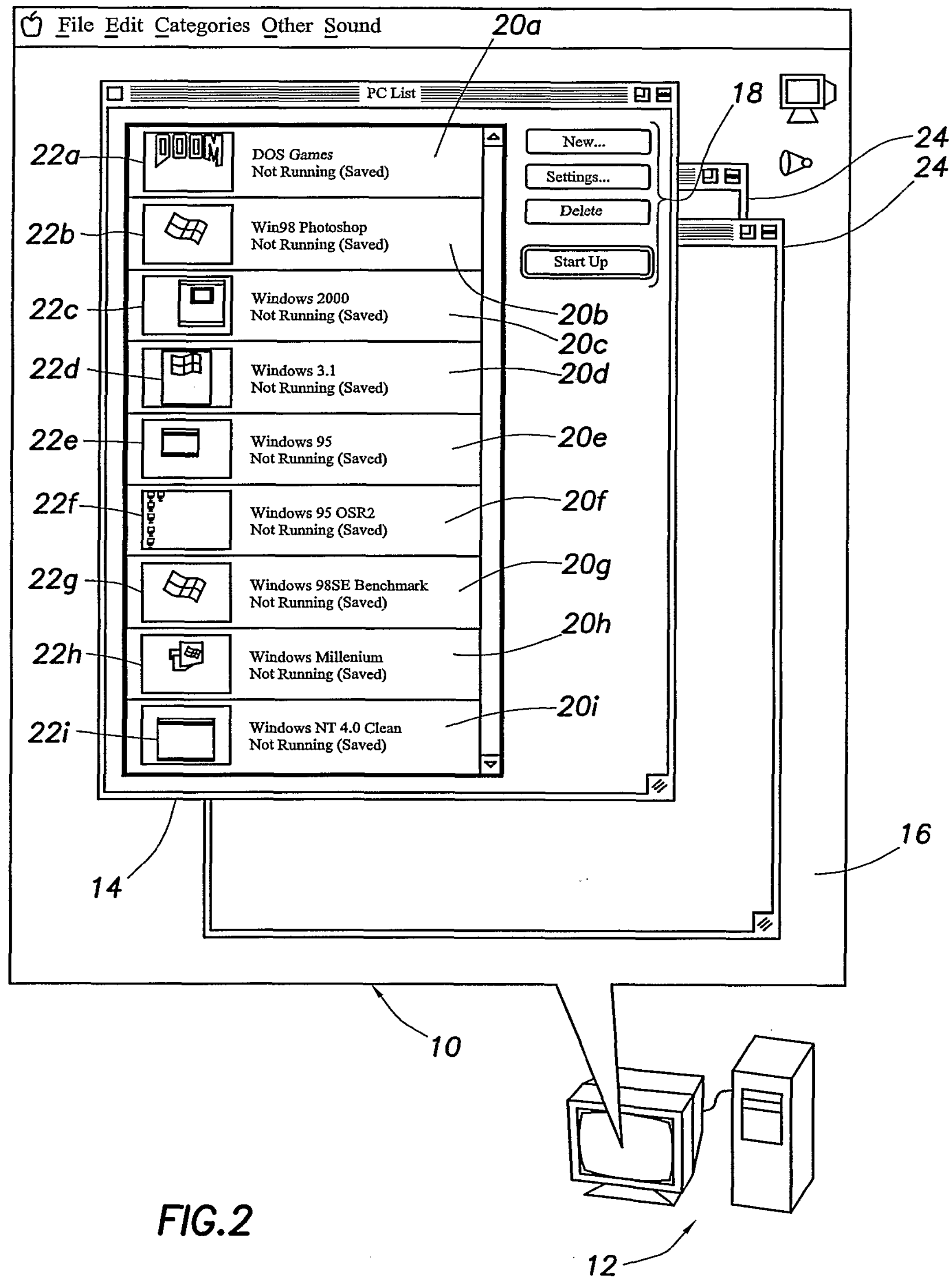


11
SOFTWARE APPLICATION
21
GUEST OPERATING SYSTEM
19
GUEST HARDWARE ARCHITECTURE
17
EMULATION PROGRAM
15
HOST OPERATING SYSTEM AND HARDWARE ARCHITECTURE
13



USER SUSPENDS VIRTUAL MACHINE
26
VIRTUAL MACHINE IMAGE SAVED TO DISK
28
CONTENTS OF VIRTUAL MACHINE VRAM INTERPRETED
30
VRAM REPRESENTATION THAT WAS SAVED TO DISK USED TO CREATE REDUCED-SIZE IMAGE
32
REDUCED-SIZE IMAGE DISPLAYED ON VIRTUAL MACHINE WINDOW LIST
34

FIG.3



File Edit Categories Other Sound
PC List
DOS Games
Not Running (Saved)
Win98 Photoshop
Not Running (Saved)
Windows 2000
Not Running (Saved)
Windows 3.1
Not Running (Saved)
Windows 95
Not Running (Saved)
Windows 95 OSR2
Not Running (Saved)
Windows 98SE Benchmark
Not Running (Saved)
Windows Millenium
Not Running (Saved)
Windows NT 4.0 Clean
Not Running (Saved)
New...
Settings...
Delete
Start Up
20a
20b
20c
20d
20e
20f
20g
20h
20i
22a
22b
22c
22d
22e
22f
22g
22h
22i
18
24
24
16
14
10
12

FIG.2

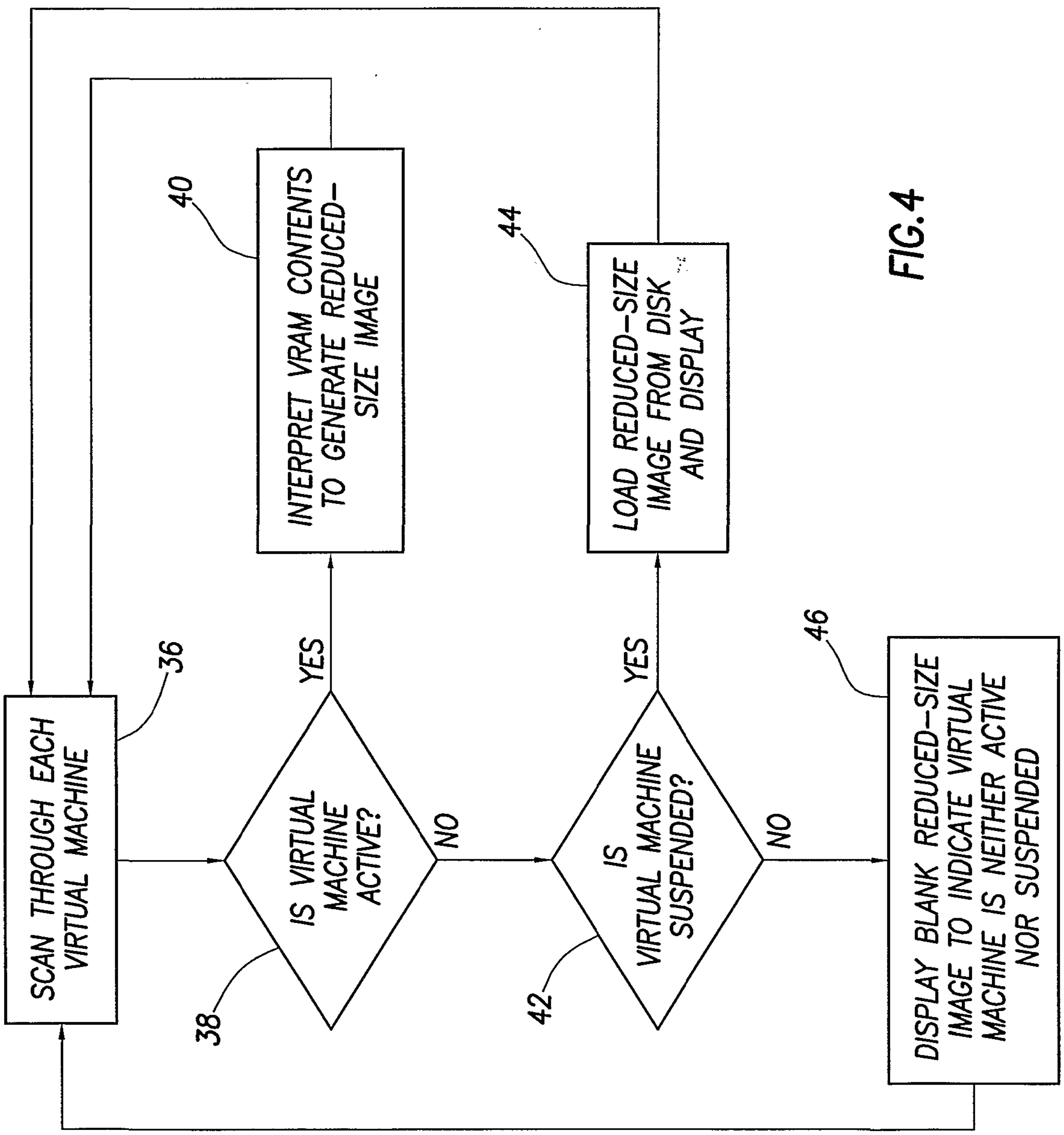

SCAN THROUGH EACH VIRTUAL MACHINE
36
IS VIRTUAL MACHINE ACTIVE?
38
YES
NO
INTERPRET VRAM CONTENTS TO GENERATE REDUCED-SIZE IMAGE
40
IS VIRTUAL MACHINE SUSPENDED?
42
YES
NO
LOAD REDUCED-SIZE IMAGE FROM DISK AND DISPLAY
44
DISPLAY BLANK REDUCED-SIZE IMAGE TO INDICATE VIRTUAL MACHINE IS NEITHER ACTIVE NOR SUSPENDED
46

FIG.4

Electronic Acknowledgement Receipt

EFS ID:	1337950
Application Number:	10939903
International Application Number:	
Confirmation Number:	5216
Title of Invention:	System for containerization of application sets
First Named Inventor/Applicant Name:	Donn Rochette
Customer Number:	27975
Filer:	Charles Edmund Wands./joellen murphy
Filer Authorized By:	Charles Edmund Wands.
Attorney Docket Number:	78802 (120-1 US)
Receipt Date:	28-NOV-2006
Filing Date:	13-SEP-2004
Time Stamp:	15:41:07
Application Type:	Utility

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)	Multi Part /.zip	Pages (if appl.)
1	Information Disclosure Statement (IDS) Filed	78802ids1.pdf	217189	no	3
Warnings:					

Information:					
This is not an USPTO supplied IDS fillable form					
2	Foreign Reference	WO2006039181.pdf	6953363	no	146
Warnings:					
Information:					
3	Foreign Reference	WO0206941A1.pdf	1039779	no	21
Warnings:					
Information:					
			Total Files Size (in bytes):	8210331	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

27975 7590 06/03/2008
ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER	
BAUM, RONALD	
ART UNIT	PAPER NUMBER
2139	

NOTIFICATION DATE	DELIVERY MODE
06/03/2008	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

creganoa@addmg.com

Office Action Summary	Application No.	Applicant(s)	
	10/939,903	ROCHETTE ET AL.	
	Examiner	**Art Unit**	
	RONALD BAUM	2139	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *13 September 2004*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-34* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-34* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *13 September 2004* is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *20080521*.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. This action is in reply to applicant's correspondence of 13 September 2004.

2. Claims 1-34 are pending for examination.

3. Claims 1-34 are rejected.

Claim Objections

4. Claim 1 is objected to because of the following informalities: the phrase "resident oh the server" is assumed to correctly be "resident on the server". Appropriate correction is required.

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

5. Claims 1-34 are rejected under 35 U.S.C. 102(b) as being anticipated by Forbes et al, U.S. Patent 6,381,742 B2 ("Forbes et al ").

6. As per claim 1; "In a system

having a plurality of servers with

operating systems that differ,

operating in

disparate computing environments,

wherein each server includes

a processor and

an operating system including

a kernel

a set of associated local system files

compatible with the processor,

a method of providing at least some of the servers in the system with

secure, executable, applications

related to a service,

wherein the applications

may be executed in

a secure environment,

wherein the applications each include

an object executable by

at least some of the different operating systems

for performing a task

related to the service,

the method comprising the steps of:

storing in memory accessible to at least some of the servers

a plurality of secure containers of application software,

each container comprising

one or more of the executable applications and

a set of associated system files

required to execute the one or more applications,

for use with a local kernel

residing permanently on

one of the servers [Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment (i.e., ' a plurality of servers … disparate computing environments … storing in memory accessible to …') using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' object executable … performing a task …') of predetermined applications (i.e., '… providing at least some of the servers … secure, executable, applications related to a service … plurality of

secure containers of application software ') and associated services across all types of code (i.e., '… a set of associated system files …') and operating systems (i.e., ' operating systems that differ '), so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.];

wherein the set of associated system files

are compatible with

a local kernel of

at least some of the plurality of different operating systems,

the containers of

application software

excluding a kernel [Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager using a distribution unit containing components

for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' compatible with … local kernel of …') of predetermined applications (i.e., '… the containers of … application software ') and associated services across all types of code and operating systems (i.e., ' plurality of different operating systems … excluding a kernel '), so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.], and

wherein some or all of the associated system files

within a container stored in memory

are utilized in place of

the associated local system files

resident oh the server prior to

said storing step [Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' are utilized in place of … associated

local system files …') of predetermined applications and associated services across all types of code and operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.].".

And further as per claim 17, this is the apparatus/system claim for the method claim 1 above, and is rejected for the same reasons provided for the claim 1 rejection; "A computing system

for performing

a plurality of tasks

each comprising

a plurality of processes

comprising:

a plurality of

secure stored containers of

associated files

accessible to, and

for execution on,

one or more servers,

each container being

mutually exclusive of the other,

such that read/write files within a container

cannot be shared with other containers,

each container of files having

its own unique identity associated therewith,

said identity comprising at least one of

an IP address,

a host name, and

a MAC address;

wherein, the plurality of files

within each of the plurality of containers comprise

one or more application programs including

one or more processes, and

associated system files

for use in executing

the one or more processes,

each container having

its own execution file associated therewith

for starting one or more applications,

in operation, each container utilizing

a kernel resident on the server and

wherein each container exclusively uses

a kernel in an underlying operation system

in which it

is running and

is absent its own kernel; and,

a run time module

for monitoring system calls from

applications associated with

one or more containers and

for providing control of

the one or more applications.”.

7. Claim 2 additionally recites the limitations that; “A method as defined in claim 1, wherein

each container has

an execution file associated therewith

for starting the one or more applications.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager distribution unit containing components for a software package and manifest file (i.e., '… container has …') that supports the installation, execution/runtime aspects (i.e., ' an execution file associated therewith … starting the one or more applications …') of predetermined applications/associated services

across all types of code and operating systems (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

8. Claim 3 additionally recites the limitations that; "A method as defined in claim 2, wherein

the execution file includes

instructions related to

an order in which

executable applications within

will be executed.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' execution file includes … order in which … will be executed ') of predetermined applications/associated services across all types of code and operating systems (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

9. Claim 4 additionally recites the limitations that; "A method as defined in claim 1 further comprising the step of

pre-identifying applications and system files

required for association with

the one or more containers

prior to said storing step.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager distribution unit containing components for a software package and manifest file (i.e., '… pre-identifying applications and system files …') that supports the installation, execution/runtime aspects (i.e., ' required for association with … one or more containers …') of predetermined applications/associated services across all types of code and operating systems (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

10. Claim 5 additionally recites the limitations that; “A method as defined in claim 2, further comprising the step of

modifying at least some of the system files

to provide an association with

a container specific identity

assigned to the container.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' modifying at least

some of the system file …') of predetermined applications /associated services across all types of code (i.e., '… provide an association with … container specific identity ') and operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

11. Claim 6 additionally recites the limitations that; "A method as defined in claim 2, comprising the step of

assigning a unique associated identity

to each of a plurality of the containers,

wherein the identity includes

at least one of

IP address,

host name, and

MAC address.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' assigning a unique associated identity …') of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes

et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

12. Claim 7 additionally recites the limitations that; “A method as defined in claim 2 further comprising the step of

modifying at least some of the system files

to define container specific mount points

associated with the container.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' modifying at least some of the system files … define container specific mount points ') of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

13. Claim 8 additionally recites the limitations that; “A method as defined in claim 1, wherein

the one or more applications and associated system files

are retrieved from a computer system having

a plurality of secure containers.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems (i.e., ' one or more applications and associated system files … a plurality of secure containers '), so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

14. Claim 9 additionally recites the limitations that; “A method as defined in claim 2, wherein

server information

related to hardware resource usage

including at least one of

CPU memory,

network bandwidth, and

disk allocation

is associated with

at least some of the containers

prior to the applications within the containers

being executed.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' server information related to hardware resource usage … associated with … prior to the applications … executed ') of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

15. Claim 10 additionally recites the limitations that; “A method as defined in claim 2, wherein

in operation when applications are executed,

applications within a container

have no access to

system files or applications in other containers or

system files within the operating system

during execution thereof

if those applications or system files

are read/write files.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' applications are executed … have no access to … system files or applications in other containers … ') of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

16. Claim 11 additionally recites the limitations that; "A method as defined in claim 2, wherein

containers include

files stored in

network file storage, and

parameters forming

descriptors of containers stored in

a separate location.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment (i.e., '… network file storage … separate location …') using a distribution unit containing components for a software package and manifest file that supports the installation,

execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

17. Claim 12 additionally recites the limitations that; "A method as defined in claim 11, further comprising the step of

merging

the files stored in network storage with

the parameters

to affect

the step of storing in claim 1.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment (i.e., '… merging … files stored in network storage … parameters …') using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

18. Claim 13 additionally recites the limitations that; "A method as defined in claim 1 further comprising the step of

associating with a container

a stored history of when processes related to applications within the container

are executed for at least one of,

tracking statistics,

resource allocation, and

for monitoring the status of the application.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' container … stored history of when processes … resource allocation ') of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

19. Claim 14 additionally recites the limitations that; "A method as defined in claim 1 comprising the step of

creating containers prior to

said step of storing containers in memory,

wherein containers are created by:

a) running an instance of a service on a server;

b) determining which files are being used; and,

c) copying applications and associated system files to memory.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation (i.e., ' creating containers prior to … containers are created by … determining which files are being used '), execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

20. Claim 15 additionally recites the limitations that; "A method as defined in claim 14 comprising the steps of:

assigning an identity to the containers

including at least one of

a unique IP address,

a unique MAC address and

an estimated resource allocation;

installing the container

on a server; and,

testing the applications and files

within the container.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types (i.e., ' assigning an identity to the containers … containers are created by … including at least one of … a unique IP address … installing the container … testing the applications and files ') as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

21. Claim 16 additionally recites the limitations that; "A method as defined in claim 1 comprising the step of

creating containers prior to

said step of storing containers in memory,

wherein a step of creating containers includes:

using a skeleton set of system files

as a container starting point and

installing applications into

that set of files.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation (i.e., ' creating containers prior to … containers includes … installing applications into '), execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

22. Claim 21 additionally recites the limitations that; "A method as defined in claim 1 further comprising the step of

installing a service on

a target server selected from

one of the plurality of servers,

wherein the step of installing the service includes the steps of:

using a graphical user interface,

associating a unique icon representing

a service with an unique icon representing

a server for hosting applications

related to the service and

for executing the service,

so as to cause the applications to be

distributed to, and

installed on

the target server.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation (i.e., ' installing a service … one of the plurality of servers … installing the service includes … a graphical user interface …'), execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

23. Claim 22 additionally recites the limitations that; “A method as defined in claim 21 wherein

the target server and

the graphical user interface

are at remote locations.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and

manifest file that supports the installation (i.e., ' target server … graphical user interface … at remote locations …'), execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

24. Claim 23 additionally recites the limitations that; "A method as defined in claim 22, wherein

the graphical user interface

is installed on a computing platform, and

wherein the computing platform

is a different computing platform than

the target server.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation (i.e., ' graphical user interface … installed on a computing platform … different computing platform '), execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

25. Claim 24 additionally recites the limitations that; "A method as defined in claim 23, wherein the step of

associating includes

the step of

relatively moving

the unique icon

representing the service to

the unique icon

representing a server.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

26. Claim 25 additionally recites the limitations that; "A method claim 21, further comprising, the step of

testing to determine if

the selected target server is

a valid computing platform,

prior to causing the applications to be

distributed to, and

installed on

the target server.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

27. Claim 26 additionally recites the limitations that; “A method according to claim 21 further comprising, the step of

creating a user account

for the service.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and

manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

28. Claim 27 additionally recites the limitations that; "A method as defined in claim 21, further comprising the step of

installing files

specific to the selected application

on the selected server.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

29. Claim 28 additionally recites the limitations that; "A method according to claim 21 further comprising the steps of

setting file access permissions

to allow a user

to access

the one of the applications

to be distributed.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

30. Claim 29 additionally recites the limitations that; “A method as defined in claim 24 further comprising the step of

starting a distributed software application.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily

extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

31. Claim 30 additionally recites the limitations that; "A method according to anyone of claims 24 further comprising the steps of

updating a console

on the selected target server

with information

indicating that the service is resident

on the selected target server.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

32. Claim 31 additionally recites the limitations that; "A method as defined in claim 1, further comprising the step of:

de-installing a service from a server,

comprising the steps of:

displaying the icon

representing the service;

displaying a the icon

representing the server on which

the service is installed; and

utilizing

the icon representing the service and

the icon representing the server

to initiating the de-installation of

the selected service from the server

on which it was installed.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime, uninstalling (i.e., '… de-installing a service from a server …') aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

33. Claim 32 additionally recites the limitations that; “A method according to claim 31 further comprising the step of

separating

icon representing the service from

the icon representing the server.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

34. Claim 33 additionally recites the limitations that; “A method according to claim 31 further comprising the step of

testing whether

the selected server

is a valid computing platform

for de-installation of the service.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network

environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime, uninstalling (i.e., '… de-installing a service from a server …') aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

35. Claim 34 additionally recites the limitations that; “A method according to claim 31 further comprising the step of

copying data file changes

specific to the service

back to a storage medium

from which the data file changes originated

prior to installation.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

36. Claim 18 additionally recites the limitations that; “A computing system as defined in claim 17, further comprising

a scheduler comprising

values related to an allotted time

in which processes within a container

may utilize predetermined resources.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

37. Claim 19 additionally recites the limitations that; “A computing system as defined in claim 18, wherein

the run time module includes

an intercepting module associated with the plurality of containers

for intercepting system calls

from any of the plurality of containers and

for providing values

alternate to values the kernel would have assigned

in response to the system calls,

so that the containers

can run independently of one another

without contention,

in a secure manner,

the values corresponding to

at least one of

the IP address,

the host name and

the MAC_Address.”.

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

38. Claim 20 additionally recites the limitations that; "A computing system as defined in claim 17, wherein

the run time module performs:

monitoring resource usage

of applications executing;

intercepting system calls to kernel mode,

made by the at least one respective application within a container,

from user mode

to kernel mode;

comparing the monitored resource usage of

the at least one respective application with

the resource limits; and,

forwarding the system calls to

a kernel

on the basis of the comparison between

the monitored resource usage and

the resource limits.".

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily

extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

Conclusion

39. Any inquiry concerning this communication or earlier communications from examiner should be directed to Ronald Baum, whose telephone number is (571) 272-3861, and whose unofficial Fax number is (571) 273-3861 and unofficial email is Ronald.baum@uspto.gov. The examiner can normally be reached Monday through Thursday from 8:00 AM to 5:30 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kristine Kincaid, can be reached at (571) 272-4063. The Fax number for the organization where this application is assigned is **571-273-8300**.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. For more information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Ronald Baum

Patent Examiner

/R. B./

Examiner, Art Unit 2139

/Kristine Kincaid/

Supervisory Patent Examiner, Art Unit 2139

Notice of References Cited	Application/Control No. 10/939,903	Applicant(s)/Patent Under Reexamination ROCHETTE ET AL.	
	Examiner RONALD BAUM	Art Unit 2139	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-6,381,742	04-2002	Forbes et al.	717/176
*	B	US-7,287,259	10-2007	Grier et al.	719/331
*	C	US-6,847,970	01-2005	Keller et al.	707/100
*	D	US-7,076,784	07-2006	Russell et al.	719/315
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

BIB DATA SHEET

CONFIRMATION NO. 5216

SERIAL NUMBER	FILING or 371(c) DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
10/939,903	09/13/2004 **RULE**	712	2139	78802 (120-1 US)

APPLICANTS
Donn Rochette, Fenton, IA;
Paul O'Leary, Kanata, CANADA;
Dean Huffman, Kanata, CANADA;

** **CONTINUING DATA** *************************
This appln claims benefit of 60/502,619 09/15/2003
and claims benefit of 60/512,103 10/20/2003

** **FOREIGN APPLICATIONS** *************************

** **IF REQUIRED, FOREIGN FILING LICENSE GRANTED** ** ** SMALL ENTITY **
11/03/2004

	STATE OR COUNTRY	SHEETS DRAWINGS	TOTAL CLAIMS	INDEPENDENT CLAIMS
Foreign Priority claimed ☐ Yes ☑ No 35 USC 119(a-d) conditions met ☐ Yes ☑ No Verified and Acknowledged /RONALD BAUM/ Examiner's Signature ☐ Met after Allowance ____ Initials	IA	17	34	2

ADDRESS

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791
UNITED STATES

TITLE

System for containerization of application sets

FILING FEE RECEIVED		
511	FEES: Authority has been given in Paper No.____________ to charge/credit DEPOSIT ACCOUNT No.____________ for following:	☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other ________________ ☐ Credit

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	10939903	ROCHETTE ET AL.
	Examiner	**Art Unit**
	RONALD BAUM	2139

SEARCHED

Class	Subclass	Date	Examiner
713	167	5/21/08	rjb
719	319	5/21/08	rjb

SEARCH NOTES

Search Notes	Date	Examiner
inventor search	5/21/08	rjb
Google(npl):server aggregate container package manifest code distribution application manager installation kernel operating seperate runtime environment	5/21/08	rjb
class 709,713,719,726,380 with text patent search: see attached EAST search history	5/21/08	rjb

INTERFERENCE SEARCH

Class	Subclass	Date	Examiner

EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	498	(713/167.ccls. or 719/319.ccls.) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/05/22 14:08
L5	26	(server and ((envelop$ container? packag$) with (application? execut$) with (install$ provision$)) and (kernel operating os unix apache linux "mac os") and ((seperate segregat$ isolat$) with (execut$ runtime run? time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/05/22 14:40
L6	32	(server and ((envelop$ container? packag$ manifest "code distribution") with (application? execut$ manager) with (install$ provision$)) and (kernel operating os unix apache linux "mac os") and ((seperate segregat$ isolat$) with (execut$ runtime run?time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/05/22 15:18
L7	15	("719"/$.ccls. "709"/$.ccls. "380"/$.ccls. "713"/$.ccls. "726"/$.ccls.) and (server and ((envelop$ container? packag$ manifest "code distribution") with (application? execut$ manager) with (install$ provision$)) and (kernel operating os unix apache linux "mac os") and ((seperate segregat$ isolat$) with (execut$ runtime run?time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/05/22 15:20

L8	47	(server and ((aggregat$ envelop$ container? packag$ manifest ((code software) adj3 distribution)) with (application? execut$ manager) with (install$ provision$)) and (kernel operating os unix apache linux "mac os" virtual) and ((mutually exclusive seperate segregat$ isolat $) with (execut$ runtime run?time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/05/22 15:33

5/ 22/ 08 3:36:32 PM
C:\ Documents and Settings\ RBaum\ My Documents\ EAST\ workspaces\ 10939903.wsp

Index of Claims	Application/Control No. 10939903	Applicant(s)/Patent Under Reexamination ROCHETTE ET AL.
	Examiner RONALD BAUM	Art Unit 2139

✓	Rejected	-	Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

CLAIM		DATE								
Final	Original	05/21/2008								
	1	✓								
	2	✓								
	3	✓								
	4	✓								
	5	✓								
	6	✓								
	7	✓								
	8	✓								
	9	✓								
	10	✓								
	11	✓								
	12	✓								
	13	✓								
	14	✓								
	15	✓								
	16	✓								
	17	✓								
	18	✓								
	19	✓								
	20	✓								
	21	✓								
	22	✓								
	23	✓								
	24	✓								
	25	✓								
	26	✓								
	27	✓								
	28	✓								
	29	✓								
	30	✓								
	31	✓								
	32	✓								
	33	✓								
	34	✓								

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE PATENT APPLICATION OF:	**ROCHETTE ET AL.**	OUR FILE NO:	**78802 120-1 US**
SERIAL NUMBER:	**10/939,903**	GROUP:	**2183**
FILED:	**SEPTEMBER 13, 2004**	CONFIRMATION NO:	**5216**
TITLE:	**SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**		

Information Disclosure Statement

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Transmitted herewith is an Information Disclosure Statement (Form PTO-1449A) in the above-captioned application with references. In accordance with current USPTO procedures published 05 AUG 2003, in 1276 OG 55, copies of the U.S. patent documents cited in the form 1449A are not attached.

Certification

______ This Information Disclosure Statement is submitted within three months of:

(i) the filing date of the above-identified U.S .National Patent application, or
(ii) the date of entry into the U.S. National Stage of the above-identified International Application, or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies.

X This Information Disclosure Statement is submitted prior to the mailing date of the first Office Action on the merits received by Applicant in the above-identified application.

______ This Information Disclosure Statement is submitted after three months from

(i) the filing date of the above-identified U.S. National Patent application, or
(ii) after three months from entry into the U.S. National Stage of the above-identified International Application; or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies; and after the mailing date of the first Office Action on the merits of the above-identified application, but prior to issuance of the earlier of any Final Action or Notice of Allowance sent in such application. The certification under 37 C.F.R. §1.97(e) is submitted separately or below, or the fee required under 37 C.F.R. §1.97(c) and § 1.17(p) is submitted herewith.

______ This Information Disclosure Statement is submitted after the earlier of the mailing date of a final rejection or Notice of Allowance sent in this application but before payment of the Issue Fee. The certification required under 37 C.F.R. § 1.97(e) is submitted separately or below. A petition to the Commissioner and the appropriate fee pursuant to §1.17(i) (1) are submitted herewith.

______ A certification under 37 C.F.R §1.97 is submitted herewith separately from this paper.

X It is hereby certified that each item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this statement.

______ No item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application or, to the knowledge of the undersigned, after making reasonable inquiry, was known to any individual designated in 37 C.F.R. §1.56(c) more than three months prior to the filing of this statement.

Respectfully submitted,

CHARLES E. WANDS
Reg. No. 25,649

Telephone: (321) 725-4760

Customer No.: 27975

CERTIFICATE OF FACSIMILE TRANSMISSION

I HEREBY CERTIFY that the foregoing correspondence has been forwarded via facsimile number 571-273-8300 to MAIL STOP AMENDMENT, COMMISSIONER FOR PATENTS, this 17 day of October 2005.

J. Kallemeres

Form PTO 1449A U.S. Department of Commerce Patent and Trademark Office Information Disclosure Statement by Applicant	ATTY. DOCKET NUMBER: 78802 120-1 US	SERIAL NUMBER: 10/939,903
	APPLICANT: ROCHETTE ET AL.	
	FILING DATE: SEPTEMBER 13, 2004	GROUP: 2183

U.S. Patent Documents

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILING DATE IF APPROPRIATE
/R.B./		2002/0174215 A1	Nov. 21, 2002	Schaefer	709	224	

Foreign Patent Documents

		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION	
							YES	NO

Other Documents (Including Author, Title, Date Pertinent Pages, Etc.)

EXAMINER /Ronald Baum/	DATE CONSIDERED 05/21/2008

EXAMINER: Initial if citation is considered, whether or not citation is in conformance with MPEP 609; draw a line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Patent Application of: Rochette et al.		**Our File:**	**78802 (120-1 US)**
Serial No:	**10/939,903**	**Group:**	**2183**
Filed:	**September 13, 2004**	**Confirmation No.**	**5216**
Title:	**SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**		

Information Disclosure Statement

EFILED
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Transmitted herewith is an Information Disclosure Statement (Form PTO-1449A) in the above-captioned application with references. In accordance with current USPTO procedures published 05 AUG 2003, in 1276 OG 55, copies of the U.S. patent documents cited in the form 1449A are not attached.

Certification

_______ This Information Disclosure Statement is submitted within three months of:

(i) the filing date of the above-identified U.S .National Patent application, or
(ii) the date of entry into the U.S. National Stage of the above-identified International Application, or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies.

__X__ This Information Disclosure Statement is submitted prior to the mailing date of the first Office Action on the merits received by Applicant in the above-identified application.

_______ This Information Disclosure Statement is submitted after three months from

(i) the filing date of the above-identified U.S. National Patent application, or
(ii) after three months from entry into the U.S. National Stage of the above-identified International Application; or
(iii) the date of entry into the U.S. National Stage of the International Application that has been assigned the above-identified U.S. Patent application number, whichever applies; and after the mailing date of the first Office Action on the merits of the above-identified application, but prior to issuance of the earlier of any Final Action or Notice of Allowance sent in such application. The certification under 37 C.F.R. §1.97(e) is submitted separately or below, or the fee required under 37 C.F.R. §1.97(c) and § 1.17(p) is submitted herewith.

_______ This Information Disclosure Statement is submitted after the earlier of the mailing date of a final rejection or Notice of Allowance sent in this application but before payment of the Issue Fee. The certification required under 37 C.F.R. § 1.97(e) is submitted separately or below. A petition to the Commissioner and the appropriate fee pursuant to §1.17(i) (1) are submitted herewith.

_______ A certification under 37 C.F.R §1.97 is submitted herewith separately from this paper.

_______ It is hereby certified that each item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application not more than three months prior to the filing of this statement.

_______ No item of information contained in this statement was cited in a communication from a foreign patent office in a counterpart foreign application or, to the knowledge of the undersigned, after making reasonable inquiry, was known to any individual designated in 37 C.F.R. §1.56(c) more than three months prior to the filing of this statement.

Respectfully submitted,

/charles edmund wands/
Reg. No: 25,649

NOV 28 2006
Date:

CUSTOMER NO. 27975

Form PTO 1449A U.S. Department of Commerce Patent and Trademark Office Information Disclosure Statement by Applicant	ATTY. DOCKET NUMBER: 78802 (120-1 US)	SERIAL NUMBER: 10/939,903
	APPLICANT: ROCHETTE ET AL.	
	FILING DATE: September 13, 2004	GROUP: 2183

U.S. Patent Documents

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILIN APPR
/R.B./		2002/0004854 A1	Jan 10, 2002	Hartley			
/R.B./		2003/0101292 A1	May 20, 2003	Fisher			

Foreign Patent Documents

		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION	
							YES	NO
/R.B./		WO 02/06941 A	Jan 24, 2002	Connectix Corporation				
/R.B./		WO 06/039181 A	Apr 13, 2006	Citrix Systems, Inc.				

Other Documents (Including Author, Title, Date Pertinent Pages, Etc.)

EXAMINER	/Ronald Baum/	DATE CONSIDERED	05/21/2008

EXAMINER: Initial if citation is considered, whether or not citation is in conformance with MPEP 609; draw a line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant

To: creganoa@addmg.com,,
From: PAIR_eOfficeAction@uspto.gov
Cc: PAIR_eOfficeAction@uspto.gov
Subject: Private PAIR Correspondence Notification for Customer Number 27975

Jun 03, 2008 07:04:53 AM

Dear PAIR Customer:

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791
UNITED STATES

The following USPTO patent application(s) associated with your Customer Number, 27975 , have new outgoing correspondence. This correspondence is now available for viewing in Private PAIR. The official date of notification of the outgoing correspondence will be indicated on the form PTOL-90 accompanying the correspondence.

Application	Attorney Docket No.
10939903	78802 (120-1 US)

To view your correspondence online or update your email addresses, please visit us anytime at https://sportal.uspto.gov/secure/myportal/privatepair. If you have any questions, please email the Electronic Business Center (EBC) at EBC@uspto.gov or call 1-866-217-9197 during the following hours:

Monday - Friday 6:00 a.m. to 12:00 a.m. Eastern Standard Time (EST)

Thank you for prompt attention to this notice,

UNITED STATES PATENT AND TRADEMARK OFFICE

PATENT APPLICATION INFORMATION RETRIEVAL SYSTEM

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of: **ROCHETTE ET AL.**	Atty. Docket No.: **78802 (120-1 US)**
Serial No. **10/939,903**	Art Unit: **2139**
Filing Date: **SEPTEMBER 13, 2004**	Examiner: **RONALD BAUM**
Confirmation No. **5216**	
For: **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**	

AMENDMENT

EFILED
Commissioner of Patents

Dear Sir:

In response to the Office Action dated June 3, 2008 please amend the above-identified application as follows:

Amendments to the Claims are reflected in the listing of claims, which begins on page 2 of this paper.

Remarks begin on page 10 of this paper.

****

In re Patent Application of:
ROCHETTE ET AL.
Serial No. **10/939,903**
Filed: **09/13/2004**

Amendments to the Claims

1. (currently amended) In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications [[may be]]are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising~~ the steps of~~:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, ~~and~~ wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server ~~prior to said storing step~~, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared

between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. (original) A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.

3. (original) A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

4. (original) A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. (currently amended) A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to ~~the~~ a particular container.

6. (original) A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. (original) A method as defined in claim 2 further comprising the step of modifying at least some of the system

files to define container specific mount points associated with the container.

8. (original) A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. (original) A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. (currently amended) A method as defined in claim 2, wherein in operation when ~~applications~~ an application residing within a container ~~are~~ is executed , ~~applications~~ said application ~~within a container have~~ has no access to system files or applications in other containers or to system files within the operating system during execution thereof ~~if those applications or system files are read/write files~~.

11. (original) A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. (original) A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.

13. (currently amended) A method as defined in claim 1 further comprising the step of associating with a ~~container~~ plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. (currently amended) A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:
a) running an instance of a service on a server;
b) determining which files are being used; and,
c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. (original) A method as defined in claim 14 comprising the steps of:
assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;
installing the container on a server; and,
testing the applications and files within the container.

16. (original) A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:
using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. (currently amended) A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure ~~stored~~ containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files ~~having~~ is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,
a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

18. (original) A computing system as defined in claim 17, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

19. (original) A computing system as defined in claim 18, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

20. (original) A computing system as defined in claim 17, wherein the run time module performs:
monitoring resource usage of applications executing;
intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;
comparing the monitored resource usage of the at least one respective application with the resource limits; and,
forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

21. (currently amended) A method as defined in claim 1 further comprising ~~the step of~~ installing a service on a target server selected from one of the plurality of servers, wherein ~~the step of~~ installing the service includes~~ the steps of~~:
using a graphical user interface, associating a unique icon representing a service with an unique icon representing a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

22. (original) A method as defined in claim 21 wherein the target server and the graphical user interface are at remote locations.

23. (original) A method as defined in claim 22, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

24. (original) A method as defined in claim 23, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

25. (original) A method claim 21, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

26. (currently amended) A method according to claim 21 further comprising~~, the step of~~ creating a user account for the service.

27. (original) A method as defined in claim 21, further comprising the step of installing files specific to the selected application on the selected server.

28. (original) A method according to claim 21 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

29. (currently amended) A method as defined in claim 24 further comprising ~~the step of~~ starting a distributed software application.

30. (currently amended) A method according to anyone of claims 24 further comprising ~~the steps of~~ updating a console on the selected target server with information indicating that the service is resident on the selected target server.

31. (currently amended) A method as defined in claim 1, further comprising ~~the step of:~~ de-installing a service from a server, comprising~~ the steps of~~:
displaying the icon representing the service;
displaying [[a]]the icon representing the server on which the service is installed;
and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

32. (currently amended) A method according to claim 31 further comprising ~~the step of~~ separating icon representing the service from the icon representing the server.

33. (currently amended) A method according to claim 31 further comprising ~~the step of~~ testing whether the selected server is a valid computing platform for de-installation of the service.

34. (currently amended) A method according to claim 31 further comprising ~~the step of~~ copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

In re Patent Application of:
ROCHETTE ET AL.
Serial No. **10/939,903**
Filed: **09/13/2004**
_________________________________/

REMARKS

Claims 1-34 are pending in this application and are rejected under 35 U.S.C. 102 (b) as being anticipated by U.S. Patent 6,381,742 B1 ("Forbes et al.").

For a claim to be anticipated, all of the limitations of the claim must be present in a single reference.

Claim 1 has been amended to include recitations which more clearly define the invention and which also distinguish Applicants' invention from the teaching of Forbes et al.

Amended Claim 1 now recites:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel **a set of associated local system files compatible** with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to

execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, ~~and~~ wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server ~~prior to said storing step~~., wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

Support for the amendments in claim 1 can be found within the specification.

For example:
At paragraph [54] the following text is found:

"In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system."

Figure 6 illustrates that system files from the underlying operating system are used independent of system files in the container. In the instant invention system files

within the container are application specific and are distinct from and system files within the OS.

Figure 9 shows that system files from the underlying operating system are used by operating system applications wherein system files from the container are used by applications associated with the container.

At paragraph [91] the following text is recited:

"Turning now to Figure 1, a system is shown having a plurality of servers 10a, 10b. Each server includes a processor and an independent operating system. Each operating system includes a kernel 12, hardware 13 in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs 11a, 11b and related system files 11c. <u>These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.</u>" (underlining added)

At paragraph [92], the following text is recited:

"Figure 2 illustrates a system in accordance with the invention in which multiple applications 21a, 21b, 21c, 21d, 21e, and 21f can execute in a secure environment on a single server. This is made possible by associating applications with secure containers 20a, 20b and 20c.

Applications are segregated and execute with their own copies of files and with a unique network identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system." (underlining added)

At paragraph [94]the following text is recited:

"The Secure application containers 20a, 20b and 20c are shown in Figure 2 in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files."

Paragraph [107] clearly teaches the relationship between files and containers:

"It can be seen from Figure 6 that each application executing associated with a container 20a or 20b, or contained within a container 20a or 20b, is able to accesses files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying

operating system of the compute platform 40 upon which they execute. Moreover, applications with a container 20a association are not able to access the files contained in another container 20b." (underlining added)

At paragraphs 112, 117, 121, 122, and 131, the following text is found which supports the amendments to Claim 1.

[112] In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.
(underlining added)

[117] Figure 3 illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.
(underlining added)

[121] Referring once again to Figure 3, applications may come from third party installation media on CDROM, package files 30, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.
(underlining added)

> [122] The second method employed to create a container file system uses an existing server, for example a computer or server already installed at a customer site. This is also shown in Figure 3. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server 32, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.
> (underlining added)

> [131] In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.
> (underlining added)

Applicants would respectfully like to point out one significant distinction between the invention of Forbes et al. and the instant invention defined in the amended claims of this invention; and more particularly amended Claim 1.

Applicants' claimed invention is defined as having a plurality of servers with operating systems that differ, operating in disparate computing environments. Applicants' claimed invention is defined as having "secure" containers of application software which include OS files that are copies or modified copies of OS files while preserving the original OS files by "not" overwriting them. These OS file copies reside with each container and are modified if necessary so as to be compatible with the applications that are to be run from any

given container. That is to say one container may be running an application that is compatible with Solaris 8, and another container may be running an application that has been written for and compatible with Solaris 10. Therefore, different containers on a single server are capable of running applications designed for different operating systems.

In contrast, Forbes et al. provide a software package manager that uses a distribution unit containing components for a software package and a manifest file that describes the distribution unit to manage the installation, execution, and uninstallation of software packages on a computer. Information in the manifest file pertaining to a software package is stored in a code store data structure upon installation of the package. The manifest file also contains information that permits the software package manager to resolve any software dependencies upon installation. The software package manager uses the code store data structure to locate the required components when the software is executed and to remove the components appropriately when the software is uninstalled.

Therefore, Forbes et al. teach a software package manager for running a software package on a system for which it was intended to run. More particularly, if the system was written or coded to be run on Windows XP, Forbes et al.'s package manager can execute on Windows XP.

Since the teaching of Forbes et al. directly instructs replacing older versions of software with newer versions by writing over older versions with newer versions, Forbes et al.

are not capable of having different versions run under the same Operating System together.

Applicants provide containers that are isolated and wherein each container has the necessary operating system files for an application to run and wherein different containers may have different versions of the same software.

Forbes et al. do not allow this. Forbes et al. teach replacement of older files with newer versions.

There is not a hint or a suggestion of having different versions of the same software on the system at the same time. There is no hint or suggestion of the provision of isolated containers of software to allow this.

Turning now to Fig. 3b of Forbes et al., the drawing (below) illustrates that contrary to Applicants' claimed invention, files are not copies in the Forbes et al. patent, they are replacements. This difference is important. Replacing an operating system file as Forbes et al. do, would not allow Applicants' claimed invention to function in the way in which it was designed.

In Forbes et al., boxes 343 and 353 show removing an older file and replacing that older file with an updated file. Boxes 359 and 361 indicate the same; "remove and replace". Since these files are removed and replaced, Forbes et al. do not use an isolated "copy" that is stored within a container. More particularly, Forbes et al. do not teach multiple

containers each having isolated copies or modified copies of system files in each container.

Not only do Forbes et al. not have the required isolation afforded by the method defined in amended Claim 1, by writing over or removing a file, the original files are not available for use in building/creating other containers. Since containers cannot share application files in Applicants' claimed invention, Forbes et al. teach a system which would not afford a plurality of isolated different containers to be created. For containers to be isolated from one another, application files cannot be shared between them. Forbes et al have a single instance installation of a single software program and are not concerned with multiple software programs in different containers.

In re Patent Application of:
ROCHETTE ET AL.
Serial No. **10/939,903**
Filed: **09/13/2004**

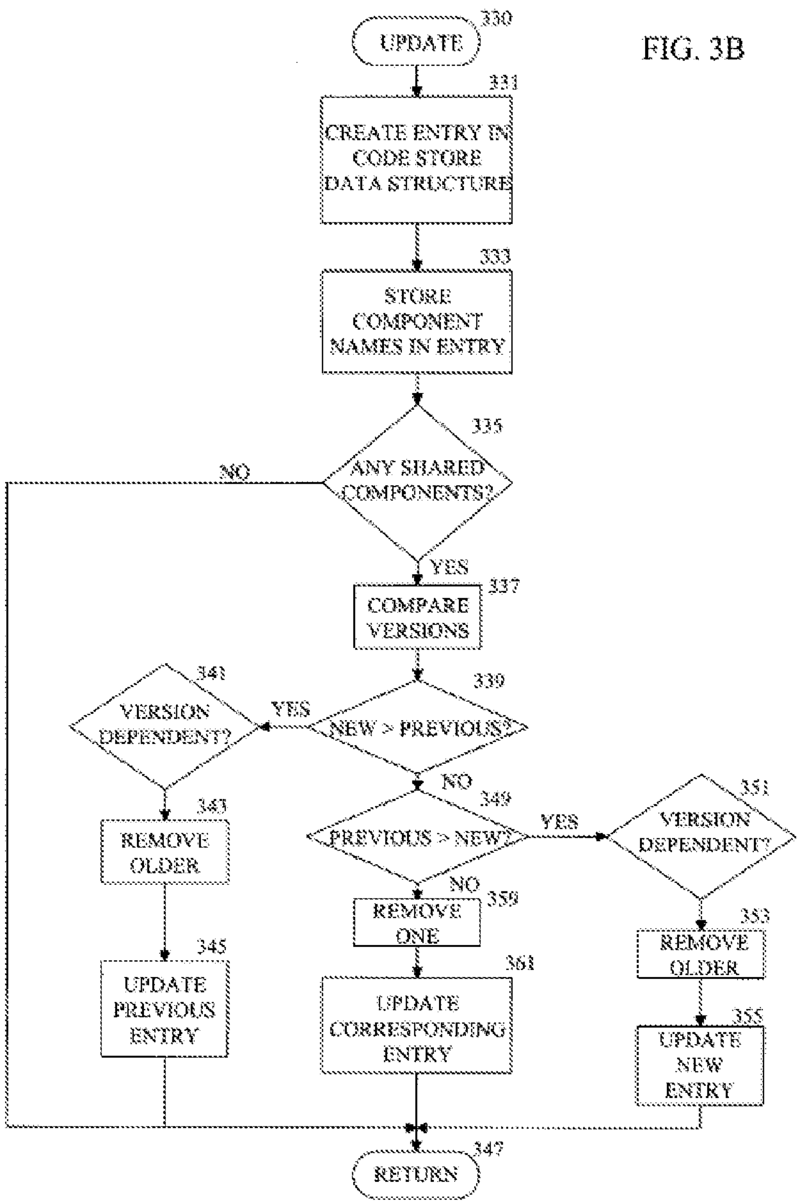


FIG. 3B

Further instances that files are not copied but are replaced in Forbes et al.'s system are found at column 2, line 66, through column 3, line 3.

> "During the installation, the package manager can optionally scan the code store data structure to determine if a component to be installed already exists on the computer and updates the code store data structure with the location of the later version of the component."
>
> Col 6 line 65 through col 7 line 2 states "Fred's Software Company also creates a "manifest" file 207 describing the CoolestApp. The CoolestApp manifest file 207 contains information about CoolestApp, including the name of the CoolestApp distribution unit 209, the version number of the software package (all components in the distribution unit 209 have the same version number in this embodiment), and the operating systems under which the CoolestApp executes. Fred's Software Company bundles the CoolestApp distribution unit 209 and manifest file 207 into a distribution unit file 205 for storage on the server 201."

The paragraph above from Forbes et al. indicates that the Operating System must be compatible with the software application. It specifies that the software application is tied to the version of the Operating System.

Thus, in Forbes et al., files being installed must be consistent with the underlying OS.

In Forbes et al., the files, and hence the application being installed, are used with the existing OS. The files are not stored into an isolated container as there are not multiple containers of application software. The files in

Forbes et al. are not isolated from the OS; and they are not able to run on an incompatible OS or version of OS.

Col 9 lines 13-18 states: "The package manager checks the name and version of the software package contained in the manifest file against the code store data structure to determine if the software package has already been installed (step 303). If so, the package manager exits (step 321)."

Thus, Forbes et al. teach that if the files exist on the system then the install exits. That is, one instance of the application exists. The files do not exist in isolated containers nor do multiple versions/copies exist nor are multiples capable of executing correctly.

Col 10, lines 20-43, states: "If the newly stored component is a newer version of the previously stored component (step 339) and the code store data structure entry for the previously installed software package that references the older version does not indicate it is version dependent (step 341), <u>the package manager removes the older version from the directory in which it is stored (step 343) and updates the code store data structure entry for the previously installed software package to point to the newer version</u> (step 345). If there is a version dependency noted, the package manager leaves the older version in the directory (step 347).

If the newly stored component is older that the previously stored component (step 349) and the software package does not indicate a version dependency (step 351), <u>the package manager removes the older version from the newly created directory (step 353) and updates the code store data</u>

structure entry for the newly installed software package to point to the newer version (step 355). As before, if there is a version dependency noted, the package manager does nothing to the older version (step 347).

If the components are the same version, the package manager chooses one to remove from its directory (step 359) and updates the corresponding entry to point to the other component (step 361)." (underlining added).

This text from Forbes et al., clearly teaches that newer versions overwrite existing older versions and so forth. In Forbes et al., a single "file" is the result. Forbes et al. do not teach that multiple copies of system files are used. There is no definition of an isolated environment in which applications execute.

In contrast, the instant invention, as defined in amended Claim 1, recites that containers have a unique root file system such that container files are distinct form other applications and from the operating system software.

Therefore, the instant invention as defined in Claim 1, creates the ability for multiple applications, including applications of the same type, to be able to execute correctly on the same Operating System. By way of example, multiple Oracle database managers are able to execute correctly on the same operating system in a container context as defined in Claim 1. This is not possible in the Forbes et al. invention.

It is significant that multiple Oracle/SQL Server/Sybase/CRM/App servers/ etc. can execute correctly on the same Operating System. The number of OS images needed, along with the associated license and management costs are greatly diminished. This enables a new and unique ability for combining, deploying and managing applications.

Claim 5 defines:

A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to ~~the~~ a particular container.

With regard to amended claim 5, Forbes et al. do not have plural containers of application software as defined in Claims 1 or 2 and do not provide an association with a container specific identity. Forbes et al. only teach having a single application not plural applications and make no mention of a container specific identity.

Claim 6 defines:

A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

The Examiner states in the Office Action that

"The teachings of Forbes et al (Forbes et al Abstract, col 2 lines 33-col. 3 line 56, figures 1-4 and associated

descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports installation, execution/runtime aspects (i.e. assigning a unique identity…") of predetermined applications/associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2 lines 38-col 3 lines 49 clearly encompasses the limitations as broadly interpreted by the examiner) suggest such limitations."

Applicants would respectfully like to point out that Claim 6 importing the limitations of amended claim 1 recites:

> In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel **a set of associated local system files compatible** with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising the steps of:
> storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications,

for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server., wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein files cannot be shared between the plurality of secure containers of application software, and wherein each of the containers each has a unique root file system that is different from the operating system's root file system, wherein each container has an execution file associated therewith for starting the one or more applications and further comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

Many of the differences in Claim 6 and the teachings of Forbes et al. have been discussed above in this response, with regard to Claim 1.

However, Forbes et al. do not teach:

- Providing secure containers;
- Containers that have copies of OS files

- Containers having a unique identity and having a unique root file system different from the root file system of the underlying OS under which the containers run; and
- assigning a unique associated identity to each of a plurality of containers, since Forbes doesn't teach "a plurality of containers" wherein each identity includes an IP address, host name or Mac Address.

The Examiner repeatedly points to col. 2 and col. 3 of Forbes et al. for teaching of the present invention. However, Applicants cannot find such teaching, and if the Examiner is of the opinion that this claim is anticipated, Applicants would be appreciative if the Examiner would indicate "specifically" where these limitations are found in Forbes et al.

In contrast, Forbes et al. teach the installation of a single software program into a system for which the software and OS are compatible. There is no hint or suggestion in Forbes et al. of storing in containers modified copies of OS system files so that software not designed for a particular OS can run.

Claim 7 defines:

A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

The Examiner has used essentially the same passages (below) from col. 2 to col. 3 to suggest that the Claim is anticipated.

In re Patent Application of:
ROCHETTE ET AL.
Serial No. **10/939,903**
Filed: **09/13/2004**

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' modifying at least some of the system files ... define container specific mount points ') of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner) suggest such limitations.

Notwithstanding, Forbes et al. do not provide the steps of:
"storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server., wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein files cannot be shared between the plurality of secure containers of

application software, and wherein each of the containers each has a unique root file system that is different from the operating system's root file system, wherein each container has an execution file associated therewith for starting the one or more applications."

Furthermore and more importantly, with reference to Claim 7, Forbes et al. do not modify at least some of the system files to define container specific mount points associated with the container. As was pointed out earlier Forbes et al. do not provide containers.

Claim 9 is "a method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed."

Claim 9 imports all of the limitations of amended Claim 1 and claim 2 from which it depends. Furthermore, Claim 9 defines within the method of Claim 2, server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

There is no such disclosure in Forbes et al. and Applicants would appreciate if the Examiner could point the Applicants to such disclosure by indicating the specific page and line number where this is taught. A careful read of col.

2 and col. 3 and the figures does not disclose the claimed limitations.

The Examiner has provided essentially the same general statement below:

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects (i.e., ' server information related to hardware resource usage ... associated with ... prior to the applications ... executed ') of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

However, the teachings of Forbes et al. in the Abstract and in Col. 2, line 33 to Col. 3, line 56, only disclose the following:

Abstract

A software package manager uses a distribution unit containing components for a software package and a manifest file that describes the distribution unit to manage the installation, execution, and uninstallation of software packages on a computer. Information in the manifest file pertaining to a software package is stored in a code store data structure upon installation of the package. The manifest file also contains information that permits the software package manager to resolve any software dependencies upon installation. The software package manager uses the code store data structure to locate the required components when the software is executed and to remove the components appropriately when the software is uninstalled.

Col. 2, line 33 to Col. 3, line 56

The above-mentioned shortcomings, disadvantages and problems are addressed by the present invention, which will be understood by reading and studying the following specification.

A software package manager uses a distribution unit containing components for a software package and a manifest file that describes the distribution unit to manage the installation, execution, and uninstallation of software packages on a computer. For installation, the package manager acquires the manifest file and parses it to learn if the software package depends on any additional components. The package manager resolves any dependencies by acquiring a distribution unit containing the needed component and installs the dependency's distribution unit as described below. Because dependencies can be nested within dependencies, the package manager recursively processes all the dependencies before finishing the installation of the software package that depends upon the additional components.

The software package manager acquires the distribution unit and extracts the components in the distribution unit into a directory on the computer. The package manager causes the operating system of the computer to install the software. The package manager then updates a code store data structure with information in the manifest file. The fields in the code store data structure contains such information as the name and version of the distribution unit, a list of the components and their location on the computer, and the source of the distribution unit. Additional fields in the code store data structure can also contain a component version, a component data type, and a digital signature if one was affixed to the distribution unit.

During the installation, the package manager can optionally scan the code store data structure to determine if a component to be installed already exists on the computer and updates the code store data structure with the location of the later version of the component.

When a user requests execution of software, the package manager uses the code store data structure to locate the appropriate components for the operating system to use. When the user requests the uninstallation of a software package, the package manager deletes the appropriate components from

the computer and updates the code store data structure accordingly.

The manifest file and distribution unit optionally are combined into a distribution unit file.

The manifest file format is common across all types of code and operating systems and easily extended to embrace new code types as they arise. The manifest file and distribution unit can be stored on all types of media from traditional magnetic and optical disks to networked servers. The distribution units for dependencies do not have to reside on the same type of media as the distribution unit or the manifest file that refers to the dependency. More than one distribution unit can be resident in a distribution unit file and a distribution unit file can contain a mixture of distribution units containing different code types.

Thus, the software package manager, the manifest file, the distribution unit and the code store data structure of the present invention solve the problems with existing distribution mechanisms. The manifest file is not particular to a particular code type or operating system and allows for the specification of nested software dependencies. Because the manifest file contains the location of the distribution units for any dependencies, the software package manager can acquire and install the dependencies without requiring manual intervention by the user. Different types of distribution units can be mixed in a distribution unit file so that a single mechanism is used to acquire and install all types of code.

The code store data structure maintained by the software package manager contains information about the installed software such as version and installation location, and is used to resolve version discrepancies among software programs that share components. The code store data structure is used by the package manager to locate necessary components when the software is executed so that a component stored in one directory can be readily shared by software programs with components in different directories. Finally, the code store data structure eases the uninstallation process by centralizing all the information about installed components.

Once again, it must be emphasized that there is no method of containerizing software applications in containers wherein the limitations of amended Claims 1, 2 and 9 are met. The Examiner recites the text above and states that the limitations of Claim 9 are present, however they are not.

Applicants believe that the amendments to Claim 1 clearly distinguish Claim 9 from the cited Forbes et al. reference.

If the Examiner is of the opinion that amended Claim 1 is anticipated by Forbes et al., Applicants would appreciate if the Examiner would specifically point out where each limitation in amended base Claim 1 is found in the text above. Furthermore, Applicants would be appreciative if the Examiner would point out where the limitations of Claim 2 and 9 can be found. It is not sufficient or instructive to Applicants to recite that the text above clearly encompasses the claimed limitations, as broadly interpreted by the Examiner. It is not clear, nor is it evident, that the claimed features are disclosed by Forbes et al.

Claim 10 has been amended to more clearly define the invention.

10. (amended) A method as defined in claim 2, wherein in operation when an application residing within a container ~~are~~ is executed , said application~~s~~ ~~within a container have~~ has no access to system files or applications in other containers or to system files within the operating system during execution thereof ~~if those applications or system files are read/write files.~~

With regard to the patentability of Claim 10, this claim imports the patentable features of Claims 1 and 2 from which it depends. Furthermore, Claim 10 defines application software within a container being isolated from other containers. Forbes et al. do not have plural containers and therefore have no isolation between containers.

Claim 13 has been amended as shown and now defines:

A method as defined in claim 1 further comprising the step of associating with a <u>plurality of </u>container<u>s</u> a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

Forbes et al. do not disclose a history of file usage or tracking of processes within an application. Once Forbes et al.'s application is executing, Forbes et al. make no suggestion or hint that there is interaction with the application. Interaction with the application is required to store container history of tracking statistics, or resource allocation, for monitoring the status of an application.

Claim 14 has been rejected as being anticipated by Forbes et al.

The Examiner has said that:

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation (i.e., ' creating containers prior to ... containers are created by ... determining which files are being used '), execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

Amended Claim 14 defines: A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

a) running an instance of a service on a server;

b) determining which files are being used; and,

c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

Claim 14 with the claim limitations of amended Claim 1 from which it depends defines:

14. (amended) In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel **a set of associated local system files compatible** with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object

executable by at least some of the different operating systems for performing a task related to the service, the method comprising the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, ~~and~~ wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server ~~prior to said storing step~~., wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers each has a unique root file system that is different from the operating system's root file system; and,

comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

a) running an instance of a service on a server;

b) determining which files are being used; and,

c) copying applications and associated system files to memory without overwriting the associated system files so as to

<u>provide a second instance of the applications and associated system files</u>.

The Examiner suggests in the paragraph below, taken from the Office Action, that Claim 14 is anticipated because:

> The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types (i.e.,' assigning an identity to the containers ... containers are created by ... including at least one of ... a unique IP address ... installing the container ... testing the applications and files ') as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

It should become readily apparent that the reference to citations taken from Forbes et al. in the paragraph above do not disclose the method steps within amended Claim 14. If the Examiner disagrees, Applicants would be appreciative if the Examiner would indicate specifically by indicating the line numbers and specific passage(s) where each claimed feature exists within the Forbes et al., patent.

Claim 15 defines: A method as defined in claim 14 comprising the steps of:
assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;
installing the container on a server; and,

testing the applications and files within the container.

Claim 15 imports the limitations of Claim 14, defined above, and further defines steps of assigning an identity to the containers. It should be noted that Forbes et al. do not have plural containers. Furthermore, assigning a unique IP address or Mac Address to plural containers is not suggested in Forbes et al. as they merely run a single instance of a software application. They do not have plural containers each having a software application each with an associated IP or Mac Address associated with a respective container where the software resides.

Forbes et al. teach that a software distribution mechanism is used with a networked system environment. The application installed, by means of Forbes et al., takes on the identity, defined as one or more of IP address, MAC, hostname, of the underlying OS. Therefore, the application installed by Forbes et al. assumes the identity supplied by the OS. The instant invention claims that an application associated with a container assumes the identity, of the IP address, or MAC, hostname, from container with which it is associated and resides in, and not from the OS.

After a careful reading of the Forbes et al. patent, it becomes evident that there is no mention of an application having an associated network identity. Forbes et al. teach that a manifest file can reside on network store; files can be accessed from a network. However, there is no teaching that an application assumes a network identity.

Claim 16 defines: A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

using a skeleton set of system files as a container starting point and installing applications into that set of files.

This claim is clearly novel as Forbes et al. do not create or provide plural containers as defined in Claim 1 or 16. In view of this, the specific implementation defined in Claim 16 is not suggested by Forbes et al.

Claim 17 has been rejected under 35 U.S.C. 102 (b) for the same reasons as Claim 1 has been rejected. Claim 17 has been amended in a similar fashion to the amendments of Claim 1.

Claim 17 defines a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file

associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,
a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

Applicants have taken the liberty of re-writing Claim 17, more clearly illustrating the features within the claim.

- A system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers,
- each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers,
- each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address;
- wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes
- <u>wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system</u>,

- each container having its own execution file associated therewith for starting one or more applications, in operation,
- each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,
- a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

➢ Forbes et al. do not have a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers; in fact Forbes et al. make no mention of containers at all, and if one were to incorrectly construe Forbes et al. as having a secure container for containing their software, this would in no way be a teaching of a plurality of secure containers having the features which follow in Claim 17.

➢ Forbes et al. do not teach that each container is mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers. Since Forbes et al. make no mention of plural containers, they clearly do not teach mutual exclusivity between containers and make no reference to an absence of file sharing of read/write files with other containers.

➢ Forbes et al. do not teach containers so they clearly do not teach or suggest that each container of files is said to have its own unique identity associated therewith,

said identity comprising at least one of an IP address, a host name, and a Mac address.

- Forbes et al. do not teach that the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, as Forbes et al. do not teach a plurality of containers.

- Forbes et al. replace and overwrite their software with a latest version so they teach away from the limitation that the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system.

- Forbes et al. do not teach plural containers wherein each container having its own execution file associated therewith for starting one or more applications, in operation. Forbes et al. only teach a single instance of a single software application.

- Forbes et al. do not teach plural containers wherein each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

- a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

After thoroughly reading the Forbes et al. cited patent and considering the invention defined in Claim 17, it becomes evident that Forbes et al. do not disclose each and every element in Claim 17. In fact, it is questionable whether Forbes et al. disclose any of the patentable elements within Claim 17. In summary, Forbes et al. do not define multiple containers and Forbes et al. do not define any isolation of files. Forbes et al. teach that more than one software distribution unit exists and that files may be shared. In Forbes et al's teaching, a distribution replaces an older version and is not isolated.

Claim 18 of the instant invention defines: A computing system as defined in claim 17, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

Forbes et al. do not have a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

In the Forbes teaching, a single application starts and there is no further interaction with the application by the package manager taught by Forbes et al. According to the invention defined by Claim 18 there is a scheduler in addition to the operating system that interacts with the application while it is executing.

Claim 19 has been rejected under 35 U.S.C. 102 (b) as being anticipated by Forbes et al.

In re Patent Application of:
ROCHETTE ET AL.
Serial No. **10/939,903**
Filed: **09/13/2004**

The Examiner has stated:

The teachings of Forbes et al (Forbes et al Abstract, col. 2,lines 33-col. 3,line 56, figures 1-4 and associated descriptions, whereas the use of a software package manager in a network environment using a distribution unit containing components for a software package and manifest file that supports the installation, execution/runtime aspects of predetermined applications /associated services across all types of code/operating systems, so as to allow easily extending new code types as they arise (e.g., Forbes et al col. 2,lines 38-col. 3,lines 49), clearly encompasses the claimed limitations as broadly interpreted by the examiner.) suggest such limitations.

Applicants have carefully reviewed the above mentioned Abstract, Col. 2 to Col. 3 and figures 1-4 and respectfully disagree with the Examiner as to a finding of anticipation. For a claim to be anticipated each and every element must be present.

Claim 19, including the limitations of amended Claim 17 defines:

- A system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers,
- each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers,
- each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address;

- wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes
- <u>wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system</u>,
- each container having its own execution file associated therewith for starting one or more applications, in operation,
- each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,
- a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications,
- wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address; and
- further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

Forbes et al. do not teach:

- Forbes et al. do not have a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers; in fact Forbes makes no mention of containers at all, and if one were to incorrectly construe Forbes as having a secure container for containing his software, this would in no way be a teaching of a plurality of secure containers having the features which follow in claim 17.

- Forbes et al. do not teach that each container is mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers. Since Forbes et al. make no mention of plural containers they clearly do not teach mutual exclusivity between containers and make no reference to an absence of file sharing of read/write files with other containers.

- Forbes et al. do not teach containers so they clearly do not teach or suggest that each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac address.

- Forbes et al. do not teach that the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, as Forbes et al. do not teach a plurality of containers.

- Forbes et al. replace and overwrite their software with a latest version so they teach away from the limitation that the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system.

- Forbes et al. do not teach plural containers wherein each container having its own execution file associated therewith for starting one or more applications, in operation. Forbes et al. only teach a single instance of a single software application.

- Forbes et al. do not teach plural containers wherein each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,
- a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

- AND more particularly Forbes et al. do not teach a run time module including an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac Address;

Since Forbes et al. do not teach a system wherein there is inception of system calls and subsequent interaction with the application after it is launched they do not teach system calls. More importantly, Forbes et al. do not include any hint or suggestion of having interception of system calls.

Having system calls advantageously provides the ability to monitor the application as it executes and change parameters that are provided to and received from the operating system, thereby altering interaction with the operating system so as to allow the application to execute in an isolated and contained environment.

Claim 20 is said to be anticipated by Forbes et al. for essentially the same reasons as Claim 17.

As Applicants have pointed out above, Claim 17 is patentable and therefore Claim 20 at least imports the patentable features of Claim 17. Furthermore, Claim 17 defines that the run time module performs:
monitoring resource usage of applications executing;
intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;
comparing the monitored resource usage of the at least one respective application with the resource limits; and,
forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

Interception of system calls is not described anywhere within the Forbes et al. patent. Forbes et al. do not monitor

after their software execution begins. Therefore they do not monitor/intercept/compare/ and forward system calls. When the Forbes et al. software is fully loaded, it executes. In the instant invention, after execution begins, the steps of monitoring/intercepting/comparing and forwarding system calls is performed. No such steps are taught by Forbes et al.

In view of the foregoing amendments to the claims and remarks, it is respectfully submitted that the instant application is now in condition for allowance.

Early and favorable reconsideration of the application would be appreciated.

Should any minor informalities need to be addressed, the Examiner is encouraged to contact the undersigned attorney at the telephone number listed below.

Please charge any shortage in fees due in connection with the filing of this paper, including Extension of Time fees, to Deposit Account No. 50-1465 and please credit any excess fees to such deposit account.

Respectfully submitted,

/CHRISTOPHER F. REGAN/
Christopher F. Regan
REG. NO. 34,906

Customer No.: 27975

Telephone: (407) 841-2330

Date: September 3, 2008

Electronic Acknowledgement Receipt

EFS ID:	3883012
Application Number:	10939903
International Application Number:	
Confirmation Number:	5216
Title of Invention:	System for containerization of application sets
First Named Inventor/Applicant Name:	Donn Rochette
Customer Number:	27975
Filer:	Christopher F. Regan/joellen murphy
Filer Authorized By:	Christopher F. Regan
Attorney Docket Number:	78802 (120-1 US)
Receipt Date:	03-SEP-2008
Filing Date:	13-SEP-2004
Time Stamp:	17:38:47
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		78802amd.pdf	324453 bd49eb94692e602b3cd241cc31e995e6bc0dab5c	yes	48

	Multipart Description/PDF files in .zip description		
	Document Description	**Start**	**End**
	Amendment - After Non-Final Rejection	1	1
	Claims	2	9
	Applicant Arguments/Remarks Made in an Amendment	10	48
Warnings:			
Information:			
	Total Files Size (in bytes):	324453	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/SB/06 (07-06)
Approved for use through 1/31/2007. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

PATENT APPLICATION FEE DETERMINATION RECORD

Substitute for Form PTO-875

Application or Docket Number	Filing Date	
10/939,903	09/13/2004	☐ To be Mailed

APPLICATION AS FILED – PART I

SMALL ENTITY ☒ OR OTHER THAN SMALL ENTITY

FOR	NUMBER FILED (Column 1)	NUMBER EXTRA (Column 2)	SMALL ENTITY RATE ($)	SMALL ENTITY FEE ($)		OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY FEE ($)
☐ BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	
☐ SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	
☐ EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X $ =		OR	X $ =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X $ =			X $ =	
☐ APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).						
☐ MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))							
			TOTAL			TOTAL	

* If the difference in column 1 is less than zero, enter "0" in column 2.

APPLICATION AS AMENDED – PART II

AMENDMENT	09/03/2008	CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	SMALL ENTITY RATE ($)	SMALL ENTITY ADDITIONAL FEE ($)		OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	* 34	Minus	** 34	= 0	X $25 =	0	OR	X $ =	
	Independent (37 CFR 1.16(h))	* 2	Minus	***3	= 0	X $105 =	0	OR	X $ =	
	☐ Application Size Fee (37 CFR 1.16(s))									
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE	**0**	OR	TOTAL ADD'L FEE	

AMENDMENT		CLAIMS REMAINING AFTER AMENDMENT (Column 1)		HIGHEST NUMBER PREVIOUSLY PAID FOR (Column 2)	PRESENT EXTRA (Column 3)	SMALL ENTITY RATE ($)	SMALL ENTITY ADDITIONAL FEE ($)		OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE ($)
	Total (37 CFR 1.16(i))	*	Minus	**	=	X $ =		OR	X $ =	
	Independent (37 CFR 1.16(h))	*	Minus	***	=	X $ =		OR	X $ =	
	☐ Application Size Fee (37 CFR 1.16(s))									
	☐ FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Legal Instrument Examiner:
/nicole c. lawrence/

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER	
BAUM, RONALD	
ART UNIT	PAPER NUMBER
2439	

DATE MAILED: 12/10/2008

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

TITLE OF INVENTION: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$755	$300	$0	$1055	03/10/2009

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE DOES NOT REFLECT A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE IN THIS APPLICATION. IF AN ISSUE FEE HAS PREVIOUSLY BEEN PAID IN THIS APPLICATION (AS SHOWN ABOVE), THE RETURN OF PART B OF THIS FORM WILL BE CONSIDERED A REQUEST TO REAPPLY THE PREVIOUSLY PAID ISSUE FEE TOWARD THE ISSUE FEE NOW DUE.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status above is to be removed, check box 5b on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check box 5a on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and 1/2 the ISSUE FEE shown above.

II. PART B - FEE(S) TRANSMITTAL, or its equivalent, must be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted. If an equivalent of Part B is filed, a request to reapply a previously paid issue fee must be clearly made, and delays in processing may occur due to the difficulty in recognizing the paper as an equivalent of Part B.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)

(Signature)

(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

TITLE OF INVENTION: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$755	$300	$0	$1055	03/10/2009

EXAMINER	ART UNIT	CLASS-SUBCLASS
BAUM, RONALD	2439	713-167000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).
   - ☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
   - ☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list
   (1) the names of up to 3 registered patent attorneys or agents OR, alternatively,
   (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.
   1 ____________________
   2 ____________________
   3 ____________________

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE

(B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent) : ☐ Individual ☐ Corporation or other private group entity ☐ Government

4a. The following fee(s) are submitted:
- ☐ Issue Fee
- ☐ Publication Fee (No small entity discount permitted)
- ☐ Advance Order - # of Copies ____________________

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**
- ☐ A check is enclosed.
- ☐ Payment by credit card. Form PTO-2038 is attached.
- ☐ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number ____________ (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)
- ☐ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27.
- ☐ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature ____________________ Date ____________________

Typed or printed name ____________________ Registration No. ____________________

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER	
BAUM, RONALD	
ART UNIT	PAPER NUMBER
2439	

DATE MAILED: 12/10/2008

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 933 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 933 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at 1-(888)-786-0101 or (571)-272-4200.

Notice of Allowability	**Application No.** 10/939,903	**Applicant(s)** ROCHETTE ET AL.	
	Examiner RONALD BAUM	**Art Unit** 2439	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *03 September 2008*.

2. ☒ The allowed claim(s) is/are *1-34*.

3. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a) ☐ All b) ☐ Some* c) ☐ None of the:
   1. ☐ Certified copies of the priority documents have been received.
   2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
   3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

4. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
   1) ☐ hereto or 2) ☐ to Paper No./Mail Date _____.
   (b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.

   Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).

6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. ☒ Notice of References Cited (PTO-892)
2. ☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
3. ☐ Information Disclosure Statements (PTO/SB/08), Paper No./Mail Date _____
4. ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
5. ☐ Notice of Informal Patent Application
6. ☐ Interview Summary (PTO-413), Paper No./Mail Date _____ .
7. ☐ Examiner's Amendment/Comment
8. ☒ Examiner's Statement of Reasons for Allowance
9. ☐ Other _____.

DETAILED ACTION

Examiner's Statement of Reasons for Allowance

1. Claims 1-34 are allowed over prior art.

2. This action is in reply to applicant's correspondence of 03 September 2008.

3. The following is an examiner's statement of reasons for the indication of allowable claimed subject matter.

4. As per claims 1 and 17 generally, prior art of record, Forbes et al, U.S. Patent 6,381,742 B2, fails to anticipate, disclose, teach or suggest alone, or in combination, at the time of the invention, the features as set forth in the claims in this application as allowed, and not necessarily as summarized and/or characterized by the examiner, whether or not as italicized, as discussed and remarked upon in the response of 03 September 2008 to office action of 03 June 2008.

Specifically, (as per claim 1, for example) prior art dealing with various aspects of systems that run multiple applications (often for a single user running unrelated applications scenarios), across independent systems/disparate workstations, utilizing multiple, often different operating systems (i.e., scenarios that require system architectures that require system virtualization with high degrees of both isolation and efficiency (e.g., Hypervisors), alternatives utilizing resource containers and security containers applied to general-purpose, time-shared operating systems exist, such as the Linux-VServer; Soltesz, S., et al, 'Container-based Operating System Virtualization:A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., Vol. 41, No. 3. (June 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=46675559), is generally known per se. Nowhere in the prior art is found collectively the *italicized* claim

elements (i.e., the various aspects of applications software not being sharable between the plurality of secure (and isolated) containers of application software, and unique root file systems different from an operating system's root file system, so as to allow for different versions of the same operating system running on the same system/server environment), at the ***time*** *of the invention*, serving to patently distinguish the invention from said prior art;

"1. In a system

having *a* ***plurality*** *of servers* with

operating systems ***that differ***,

operating in

disparate *computing environments*,

wherein each server includes

a processor and

an operating system including

a kernel

a set of associated local system files

compatible with the processor,

a method of providing at least some of the servers in the system with

secure, executable, ***applications***

related to ***a service***,

wherein the applications

are executed in

a secure environment,

wherein the *applications* ***each*** *include*

an object *executable by*

at least **some of the different** *operating systems*

for performing a task

related to the service,

the method comprising:

storing in memory ***accessible*** *to at least some of the servers*

a *plurality of secure containers of application software*,

each *container* comprising

one or more of the *executable applications* and

a set of *associated system files*

required to execute the one or more *applications*,

for use with ***a local kernel***

residing ***permanently on***

one of the servers;

wherein the set of *associated system files*

are *compatible with*

a local kernel of

at least *some of the* **plurality** *of* **different** *operating systems*,

the ***containers of***

application software

excluding a kernel,

wherein some or all of the *associated* ***system files***

within *a container* stored in memory

are *utilized* ***in place of***

the associated ***local system files***

that ***remain resident*** *on the server*,

wherein *said associated system files*

utilized in place of

the associated *local system files*

are ***copies or modified copies*** *of*

the *associated* ***local*** *system files*

that ***remain resident on the server***, and

wherein the ***application*** *software*

cannot be shared between

the plurality of ***secure*** *containers of* ***application software,*** and

wherein ***each*** *of the containers has*

a unique *root file system*

that is ***different*** *from*

an operating system's root file system."

5. Dependent claims 2-16 and 18-34 are allowable by virtue of their dependencies.

Conclusion

6. Any inquiry concerning this communication or earlier communications from examiner should be directed to Ronald Baum, whose telephone number is (571) 272-3861, and whose unofficial Fax number is (571) 273-3861 and unofficial email is Ronald.baum@uspto.gov. The examiner can normally be reached Monday through Thursday from 8:00 AM to 5:30 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kambiz Zand, can be reached at (571) 272-3811. The Fax number for the organization where this application is assigned is **571-273-8300**.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. For more information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Ronald Baum

Patent Examiner

/R. B./

Examiner, Art Unit 2439

/Kambiz Zand/

Supervisory Patent Examiner, Art Unit 2434

Notice of References Cited	Application/Control No. 10/939,903	Applicant(s)/Patent Under Reexamination ROCHETTE ET AL.	
	Examiner RONALD BAUM	Art Unit 2439	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-			
	B	US-			
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	Soltesz, S., et al, 'Container-based Operating System Virtualization:A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., Vol. 41, No. 3. (June 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=4667
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	524	(713/167.ccls. 719/319. ccls.) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/12/02 16:13
L2	26	(server and ((envelop$ container? packag$) with (application? execut$) with (install$ provision$)) and (kernel operating os unix apache linux "mac os") and ((seperate segregat$ isolat$) with (execut$ runtime run? time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/12/02 16:14
L3	15	("719"/$.ccls. "709"/$. ccls. "380"/$.ccls. "713"/$. ccls. "726"/$.ccls.) and (server and ((envelop$ container? packag$ manifest "code distribution") with (application? execut$ manager) with (install$ provision$)) and (kernel operating os unix apache linux "mac os") and ((seperate segregat$ isolat $) with (execut$ runtime run?time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/12/02 16:14
L4	48	(server and ((aggregat$ envelop$ container? packag$ manifest ((code software) adj3 distribution)) with (application? execut$ manager) with (install$ provision$)) and (kernel operating os unix apache linux "mac os" virtual) and ((mutually exclusive seperate segregat$ isolat $) with (execut$ runtime run?time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/12/02 16:15

L5	2	((server and ((aggregat$ envelop$ container? packag$ manifest ((code software) adj3 distribution)) with (application? execut$ manager) with (install$ provision$)) and (kernel operating os unix apache linux "mac os" virtual) and ((mutually exclusive seperate segregat$ isolat $) with (execut$ runtime run?time environment)))).clm.	US-PGPUB	OR	OFF	2008/12/02 16:16
S5	66555	("709"/$.ccls. or "380"/$.ccls. or "713"/$.ccls. or "726"/$.ccls.) and @ad<"20021018"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/01/30 14:49
S15	498	(713/167.ccls. or 719/319.ccls.) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/05/22 14:08
S19	26	(server and ((envelop$ container? packag$) with (application? execut$) with (install$ provision$)) and (kernel operating os unix apache linux "mac os") and ((seperate segregat$ isolat$) with (execut$ runtime run?time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/05/22 14:40
S20	32	(server and ((envelop$ container? packag$ manifest "code distribution") with (application? execut$ manager) with (install$ provision$)) and (kernel operating os unix apache linux "mac os") and ((seperate segregat$ isolat $) with (execut$ runtime run?time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/05/22 15:18

S21	15	("719"/$.ccls. "709"/$.ccls. "380"/$.ccls. "713"/$.ccls. "726"/$.ccls.) and (server and ((envelop$ container? packag$ manifest "code distribution") with (application? execut$ manager) with (install$ provision$)) and (kernel operating os unix apache linux "mac os") and ((seperate segregat$ isolat$) with (execut$ runtime run?time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/05/22 15:20
S22	47	(server and ((aggregat$ envelop$ container? packag$ manifest ((code software) adj3 distribution)) with (application? execut$ manager) with (install$ provision$)) and (kernel operating os unix apache linux "mac os" virtual) and ((mutually exclusive seperate segregat$ isolat$) with (execut$ runtime run?time environment))) and @ad<"20030915"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2008/05/22 15:33

12/ 2/ 08 4:39:31 PM
C:\ Documents and Settings\ RBaum\ My Documents\ EAST\ workspaces\ 10939903.wsp

Index of Claims	Application/Control No. 10939903	Applicant(s)/Patent Under Reexamination ROCHETTE ET AL.
	Examiner RONALD BAUM	Art Unit 2439

✓	Rejected	-	Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

CLAIM		DATE								
Final	Original	05/21/2008	12/02/2008							
1	1	✓	=							
2	2	✓	=							
3	3	✓	=							
4	4	✓	=							
5	5	✓	=							
6	6	✓	=							
7	7	✓	=							
8	8	✓	=							
9	9	✓	=							
10	10	✓	=							
11	11	✓	=							
12	12	✓	=							
13	13	✓	=							
14	14	✓	=							
15	15	✓	=							
16	16	✓	=							
31	17	✓	=							
32	18	✓	=							
33	19	✓	=							
34	20	✓	=							
17	21	✓	=							
18	22	✓	=							
19	23	✓	=							
20	24	✓	=							
23	25	✓	=							
24	26	✓	=							
25	27	✓	=							
26	28	✓	=							
21	29	✓	=							
22	30	✓	=							
27	31	✓	=							
28	32	✓	=							
29	33	✓	=							
30	34	✓	=							

Issue Classification	Application/Control No.	Applicant(s)/Patent Under Reexamination
	10939903	ROCHETTE ET AL.
	Examiner	**Art Unit**
	RONALD BAUM	2439

ORIGINAL		INTERNATIONAL CLASSIFICATION									
CLASS	SUBCLASS	CLAIMED					NON-CLAIMED				
713	167	G	0	6	F	3 / 00 (2006.01.01)	G	0	6	F	3 / 00 (2006.01.01)

CROSS REFERENCE(S)

CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)				
713	164				
709	248	214			
719	319				

☐ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original
1	1	31	17	29	33										
2	2	32	18	30	34										
3	3	33	19												
4	4	34	20												
5	5	17	21												
6	6	18	22												
7	7	19	23												
8	8	20	24												
9	9	23	25												
10	10	24	26												
11	11	25	27												
12	12	26	28												
13	13	21	29												
14	14	22	30												
15	15	27	31												
16	16	28	32												

/RONALD BAUM/ Examiner.Art Unit 2439 (Assistant Examiner)	20081202 (Date)	**Total Claims Allowed:** 34	
/KAMBIZ ZAND/ Supervisory Patent Examiner.Art Unit 2434 (Primary Examiner)	12/04/2008 (Date)	O.G. Print Claim(s) 1	O.G. Print Figure 1

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	10939903	ROCHETTE ET AL.
	Examiner	**Art Unit**
	RONALD BAUM	2139

SEARCHED

Class	Subclass	Date	Examiner
713	167	12/2/08	rjb
719	319	12/2/08	rjb

SEARCH NOTES

Search Notes	Date	Examiner
inventor search	12/2/08	rjb
Google(npl):server aggregate container package manifest code distribution application manager installation kernel operating seperate runtime environment	12/2/08	rjb
class 709,713,719,726,380 with text patent search: see attached EAST search history	12/2/08	rjb
claim text interference patent search: see attached EAST search history	12/2/08	rjb
consult primary (C. Laforgia)	12/2/08	rjb

INTERFERENCE SEARCH

Class	Subclass	Date	Examiner
none	((server and ((aggregat$ envelop$ container? packag$ manifest ((code software) adj3 distribution)) with (application? execut$ manager) with (install$ provision$)) and (kernel operating os unix apache linux "mac os" virtual) and ((mutually exclusive seperate segregat$ isolat$) with (execut$ runtime run?time environment)))).clm.	12/2/08	rjb

Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors

Stephen Soltesz
Dept. of Computer Science
Princeton University
Princeton, New Jersey 08540
soltesz@cs.princeton.edu

Herbert Pötzl
Linux-VServer Maintainer
Laaben, Austria
herbert@13thfloor.at

Marc E. Fiuczynski
Dept. of Computer Science
Princeton University
Princeton, New Jersey 08540
mef@cs.princeton.edu

Andy Bavier
Dept. of Computer Science
Princeton University
Princeton, New Jersey 08540
acb@cs.princeton.edu

Larry Peterson
Dept. of Computer Science
Princeton University
Princeton, New Jersey 08540
llp@cs.princeton.edu

ABSTRACT
Hypervisors, popularized by Xen and VMware, are quickly becoming commodity. They are appropriate for many usage scenarios, but there are scenarios that require system virtualization with high degrees of both *isolation* and *efficiency*. Examples include HPC clusters, the Grid, hosting centers, and PlanetLab. We present an alternative to hypervisors that is better suited to such scenarios. The approach is a synthesis of prior work on *resource containers* and *security containers* applied to general-purpose, time-shared operating systems. Examples of such container-based systems include Solaris 10, Virtuozzo for Linux, and Linux-VServer. As a representative instance of container-based systems, this paper describes the design and implementation of Linux-VServer. In addition, it contrasts the architecture of Linux-VServer with current generations of Xen, and shows how Linux-VServer provides comparable support for isolation and superior system efficiency.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures; D.4.8 [**Operating Systems**]: Performance—*Measurements, Operational analysis*

General Terms
Performance Measurement Design

Keywords
Linux-VServer Xen virtualization container hypervisor operating system alternative

EuroSys'07, March 21–23, 2007, Lisboa, Portugal.

1. INTRODUCTION
Operating system designers face a fundamental tension between isolating applications and enabling sharing among them—to simultaneously support the illusion that each application has the physical machine to itself, yet let applications share objects (e.g., files, pipes) with each other. Today's commodity operating systems, designed for personal computers and adapted from earlier time-sharing systems, typically provide a relatively weak form of isolation (the process abstraction) with generous facilities for sharing (e.g., a global file system and global process ids). In contrast, hypervisors strive to provide full isolation between virtual machines (VMs), providing no more support for sharing between VMs than the network provides between physical machines.

The workload requirements for a given system will direct users to the point in the design space that requires the least trade-off. For instance, workstation operating systems generally run multiple applications on behalf of a single user, making it natural to favor sharing over isolation. On the other hand, hypervisors are often deployed to let a single machine host multiple, unrelated applications, which may run on behalf of independent organizations, as is common when a data center consolidates multiple physical servers. The applications in such a scenario have no need to share information. Indeed, it is important they have no impact on each other. For this reason, hypervisors heavily favor full isolation over sharing. However, when each virtual machine is running the same kernel and similar operating system distributions, the degree of isolation offered by hypervisors comes at the cost of efficiency relative to running all applications on a single kernel.

A number of emerging usage scenarios—such as HPC clusters, Grid, web/db/game hosting organizations, distributed hosting (e.g., PlanetLab, Akamai, Amazon EC2)—benefit from virtualization techniques that isolate different groups of users and their applications from one another. What these usage scenarios share is the need for efficient use of system resources, either in terms of raw performance for a single or small number of VMs, or in terms of sheer scalability of

concurrently active VMs.

This paper describes a virtualization approach designed to enforce a high degree of isolation between VMs while maintaining efficient use of system resources. The approach synthesizes ideas from prior work on *resource containers* [2, 14] and *security containers* [7, 19, 12, 25] and applies it to general-purpose, time-shared operating systems. Indeed, variants of such container-based operating systems are in production use today—e.g., Solaris 10 [19], Virtuozzo [23], and Linux-VServer [11].

The paper makes two contributions. First, this is the first thorough description of the techniques used by Linux-VServer for an academic audience (henceforth referred to as just "VServer"). We choose VServer as the representative instance of the container-based system for several reasons: 1) it is open source, 2) it is in production use, and 3) because we have real data and experience from operating 700+ VServer-enabled machines on PlanetLab [17].

Second, we contrast the architecture of VServer with a recent generation of Xen, which has changed drastically since its original design was described by Barham et al. [3]. In terms of performance, the two solutions are equal for CPU bound workloads, whereas for I/O centric (server) workloads VServer makes more efficient use of system resources and thereby achieves better overall performance. In terms of scalability, VServer far surpasses Xen in usage scenarios where overbooking of system resources is required (e.g., PlanetLab, managed web hosting, etc), whereas for reservation based usage scenarios involving a small number of VMs VServer retains an advantage as it inherently avoids duplicating operating system state.

The next section presents a motivating case for container based systems. Section 3 presents container-based techniques in further detail, and describes the design and implementation of VServer. Section 4 reproduces benchmarks that have become familiar metrics for Xen and contrasts those with what can be achieved by VServer. Section 5 describes the kinds of interference observed between VMs. Finally, Section 6 offers some concluding remarks.

2. MOTIVATION

Virtual machine technologies are the product of diverse groups with different terminology. To ease the prose, we settle on referring to the isolated execution context running on top of the underlying system providing virtualization as a *virtual machine* (VM), rather than a *domain*, *container*, *applet*, *guest*, etc.. There are a variety of VM architectures ranging from the hardware (e.g., Intel's VT.) up the full software including hardware abstraction layer VMs (e.g., Xen, VMware ESX), system call layer VMs (e.g., Solaris, VServer), hosted VMs (e.g., VMware GSX), emulators (e.g, QEMU), high-level language VMs (e.g., Java), and application-level VMs (e.g., Apache virtual hosting). Within this wide range, we focus on comparing hypervisor technology that isolate VMs at the hardware abstraction layer with container-based operating system (COS) technology that isolate VMs at the system call/ABI layer.

The remainder of this section first outlines the usage scenarios of VMs to set the context within which we compare and contrast the different approaches to virtualization. We then make a case for container-based virtualization with these usage scenarios.

2.1 Usage Scenarios

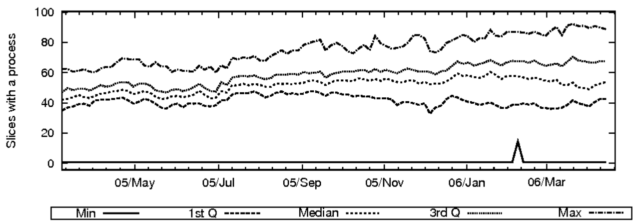
There are many innovative ideas that exploit VMs to secure work environments on laptops, detect virus attacks in real-time, determine the cause of computer break-ins, and debug difficult to track down system failures. Today, VMs are predominantly used by programmers to ease software development and testing, by IT centers to consolidate dedicated servers onto more cost effective hardware, and by traditional hosting organizations to sell virtual private servers. Other emerging, real-world scenarios for which people are either considering, evaluating, or actively using VM technologies include HPC clusters, the Grid, and distributed hosting organizations like PlanetLab and Amazon EC2. This paper focuses on these three emerging scenarios, for which efficiency is paramount.

Compute farms, as idealized by the Grid vision and typically realized by HPC clusters, try to support many different users (and their application's specific software configurations) in a batch-scheduled manner. While compute farms do not need to run many concurrent VMs (often just one per physical machine at a time), they are nonetheless very sensitive to raw performance issues as they try to maximize the number of jobs they can push through the overall system per day. As well, experience shows that most software configuration problems encountered on compute farms are due to incompatibilities of the system software provided by a specific OS distribution, as opposed to the kernel itself. Therefore, giving users the ability to use their own distribution or specialized versions of system libraries in a VM would resolve this point of pain.

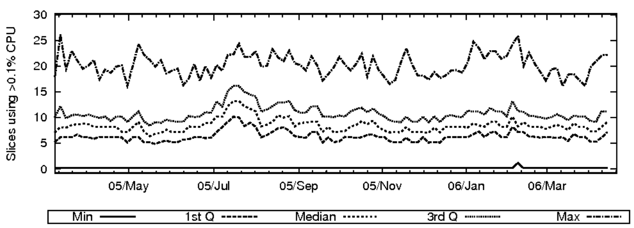
In contrast to compute farms, hosting organizations tend to run many copies of the same server software, operating system distribution, and kernels in their mix of VMs. In for-profit scenarios, hosting organizations seek to benefit from an economy of scale and need to reduce the marginal cost per customer VM. Such hosting organizations are sensitive to issues of efficiency as they try to carefully oversubscribe their physical infrastructure with as many VMs as possible, without reducing overall quality of service. Unfortunately, companies are reluctant to release just how many VMs they operate on their hardware.

Fortunately, CoMon [24]—one of the performance-monitoring services running on PlanetLab—publicly releases a wealth of statistics relating to the VMs operating on PlanetLab. PlanetLab is a non-profit consortium whose charter is to enable planetary-scale networking and distributed systems research at an unprecedented scale. Research organizations join by dedicating at least two machines connected to the Internet to PlanetLab. PlanetLab lets researchers use these machines, and each research project is placed into a separate VM per machine (referred to as a slice). PlanetLab supports a workload consisting of a mix of one-off experiments and long-running services with its slice abstraction.

CoMon classifies a VM as *active* on a node if it contains



(a) Active slices, by quartile



(b) Live slices, by quartile

Figure 1: Active and live slices on PlanetLab

a process, and *live* if, in the last five minutes, it used at least 0.1% (300ms) of the CPU. Figure 1 (reproduced from [17]) shows, by quartile, the distribution of active and live VMs across PlanetLab during the past year. Each graph shows five lines; 25% of PlanetLab nodes have values that fall between the first and second lines, 25% between the second and third, and so on. We note that, in any five-minute interval, it is not unusual to see 10-15 live VMs and 60 active VMs on PlanetLab. At the same time, PlanetLab nodes are PC-class boxes; the average PlanetLab node has a 2GHz CPU and 1GB of memory. Any system that hosts such a workload on similar hardware must be concerned with overall efficiency–i.e., both performance and scalability–of the underlying virtualization technology.

2.2 Case for COS Virtualization

The case for COS virtualization rests on the observation that it is acceptable in some real-world scenarios to trade *isolation* for *efficiency*. Sections 4 and 5 demonstrate quantitatively that a COS (VServer) is more efficient than a well designed hypervisor (Xen). So, the question remains: what must be traded to get that performance boost?

Efficiency can be measured in terms of overall performance (throughput, latency, etc) and/or scalability (measured in number of concurrent VMs) afforded by the underlying VM technology. Isolation is harder to quantify than efficiency. A system provides full isolation when it supports a combination of fault isolation, resource isolation, and security isolation. As the following discussion illustrates, there is significant overlap between COS- and hypervisor-based technologies with respect to these isolation characteristics.

Fault isolation reflects the ability to limit a buggy VM from affecting the stored state and correct operation of other VMs. Complete fault isolation between VMs requires there to be no sharing of code or data. In COS- and hypervisor-based systems, the VMs themselves are fully fault isolated from each other using address spaces. The only code and data shared among VMs is the underlying system providing virtualization—i.e., the COS or hypervisor. Any fault in this shared code base can cause the whole system to fail.

Arguably the smaller code base of a hypervisor–Xen for x86 consists of roughly 80K lines of code–naturally eases the engineering task to ensure its reliability. While this may be true, a functioning hypervisor-based system also requires a *host* VM that authorizes and multiplexes access to devices. The host VM typically consists of a fully fledged Linux (millions of lines of code) and therefore is the weak link with respect to fault isolation—i.e., a fault in the host VM could cause the whole system to fail. Fraser et al. [6] propose to mitigate this problem by isolating device drivers into independent driver domains (IDDs).

While the overall Linux kernel is large due to the number of device drivers, filesystems, and networking protocols, at its core it is less than 140K lines. To improve resilience to faults (usually stemming from drivers), Swift et al. [22] propose to isolate device drivers into IDDs within the Linux kernel using their Nooks technology. Unfortunately, there exists no study that directly compares Xen+IDD and Linux+Nooks quantitatively with respect to their performance.

With respect to fault isolation, if we accept that various subsystems such as device drivers, filesystems, networking protocols, etc. are rock solid, then the principle difference between hypervisor- and COS-based systems is in the interface they expose to VMs. Any vulnerability exposed by the implementation of these interfaces may let a fault from one VM leak to another. For hypervisors there is a narrow interface to events and virtual device, whereas for COSs there is the wide system call ABI. Arguably it is easier to verify the narrow interface, which implies that the interface exposed by hypervisors are more likely to be correct.

Resource isolation corresponds to the ability to account for and enforce the resource consumption of one VM such that guarantees and fair shares are preserved for other VMs. Undesired interactions between VMs are sometimes called *cross-talk* [9]. Providing resource isolation generally involves careful allocation and scheduling of physical resources (e.g., cycles, memory, link bandwidth, disk space), but can also be influenced by sharing of logical resources, such as file descriptors, ports, PIDs, and memory buffers. At one extreme, a virtualized system that supports resource reservations might guarantee that a VM will receive 100 million cycles per second (Mcps) and 1.5Mbps of link bandwidth, independent of any other applications running on the machine. At the other extreme, a virtualized system might let VMs obtain cycles and bandwidth on a demand-driven (best-effort) basis. Many hybrid approaches are also possible: for instance, a system may enforce fair sharing of resources between classes of VMs, which lets one overbook available resources while preventing starvation in overload scenarios. The key point is that both hypervisors and COSs incorporate sophisticated resource schedulers to avoid or minimize crosstalk.

Security isolation refers to the extent to which a virtualized system limits access to (and information about) logical objects, such as files, virtual memory addresses, port numbers, user ids, process ids, and so on. In doing so, security isolation promotes (1) *configuration independence*, so that global names (e.g., of files, SysV Shm keys, etc) selected by one VM cannot conflict with names selected by another VM; and (2) *safety*, such that when global namespaces are shared, one VM is not able to modify data and code belonging to another VM, thus diminishing the likelihood that a compromise to one VM affects others on the same machine. A virtualized system with complete security isolation does not reveal the names of files or process ids belonging to another VM, let alone let one VM access or manipulate such objects. In contrast, a virtualized system that supports partial security isolation might support a shared namespace (e.g., a global file system), augmented with an access control mechanism that limits the ability of one VM to manipulate the objects owned by another VM. As discussed later, some COSs opt to apply access controls on shared namespaces, as opposed to maintaining autonomous and opaque namespaces via contextualization, in order to improve performance. In such a partially isolated scheme, information leaks are possible, for instance, allowing unauthorized users to potentially identify in-use ports, user names, number of running processes, etc. But, both hypervisors and COSs can hide logical objects in one VM from other VMs to promote both configuration independence and system safety.

Discussion: VM technologies are often embraced for their ability to provide strong isolation as well as other value-added features. Table 1 provides a list of popular value-added features that attract users to VM technologies, which include abilities to run multiple kernels side-by-side, have administrative power (i.e., root) within a VM, checkpoint and resume, and migrate VMs between physical hosts.

Features	Hypervisor	Containers
Multiple Kernels	✓	×
Administrative power (root)	✓	✓
Checkpoint & Resume	✓	✓ [15,23,18]
Live Migration	✓	✓ [23,18]
Live System Update	×	✓ [18]

Table 1: Feature comparison of hypervisor- and COS-based systems

Since COSs rely on a single underlying kernel image, they are of course not able to run multiple kernels like hypervisors can. As well, the more low-level access that is desired by users, such as the ability to load a kernel module, the more code is needed to preserve isolation of the relevant system. However, some COSs can support the remaining features. The corresponding references are provided in Table 1. In fact, at least one solution supporting COS-based VM migration goes a step further than hypervisor-based VM migration: it enables VM migration from one kernel *version* to another. This feature lets systems administrators do a Live System Update on a running system, e.g., to release a new kernel with bug/security fixes, performance enhancements, or new features, without needing to reboot the VM. Kernel version migration is possible because COS-based solutions have explicit knowledge of the dependencies that processes within a VM have to in-kernel structures [18].

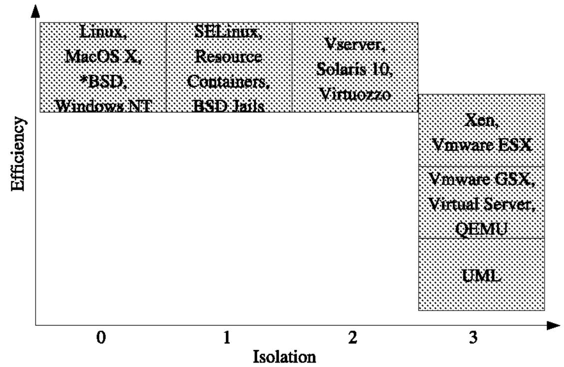


Figure 2: Summary of existing hypervisor- and COS-based technology

Figure 2 summarizes the state-of-the-art in VM technology along the efficiency and isolation dimensions. The x-axis counts how many of the three different kinds of isolation are supported by a particular technology. The y-axis is intended to be interpreted qualitatively rather than quantitatively; as mentioned, later sections will focus on presenting quantitative results.

The basic observation made in the figure is, to date, there is no VM technology that achieves the ideal of maximizing both efficiency and isolation. We argue that for usage scenarios where efficiency trumps the need for full isolation, a COS such as VServer hits the sweet spot within this space. Conversely, for scenarios where full isolation is required, a hypervisor is best. Finally, since the two technologies are not mutually exclusive, one can run a COS in a VM on a hypervisor when appropriate.

3. CONTAINER-BASED OS APPROACH

This section provides an overview of container-based systems, describes the general techniques used to achieve isolation, and presents the mechanisms with which VServer implements these techniques.

3.1 Overview

A container-based system provides a shared, virtualized OS image consisting of a root file system, a (safely shared) set of

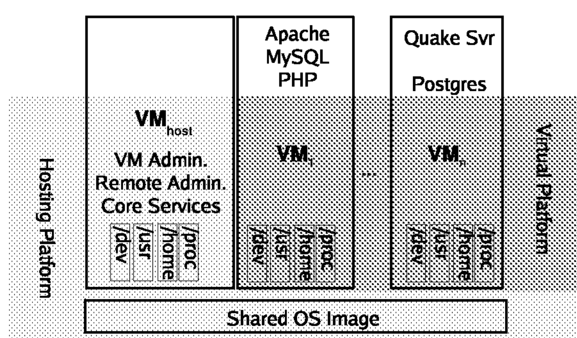


Figure 3: COS Overview

system libraries and executables. Each VM can be booted, shut down, and rebooted just like a regular operating system. Resources such as disk space, CPU guarantees, memory, etc. are assigned to each VM when it is created, yet often can be dynamically varied at run time. To applications and the user of a container-based system, the VM appears just like a separate host. Figure 3 schematically depicts the design.

As shown in the figure, there are three basic platform groupings. The hosting platform consists essentially of the shared OS image and a privileged *host VM*. This is the VM that a system administrator uses to manage other VMs. The virtual platform is the view of the system as seen by the *guest VMs*. Applications running in the guest VMs work just as they would on a corresponding non-container-based OS image. At this level, there is little difference between a container and hypervisor based system. However, they differ fundamentally in the techniques they use to implement isolation between VMs.

Figure 4 illustrates this by presenting a taxonomic comparison of their security and resource isolation schemes. As shown in the figure, the COS approach to security isolation directly involves internal operating system objects (PIDs, UIDs, Sys-V Shm and IPC, Unix ptys, and so on). The basic techniques used to securely use these objects involve: (1) separation of name spaces (contexts), and (2) access controls (filters). The former means that global identifiers (e.g., process ids, SYS V IPC keys, user ids, etc.) live in completely different spaces (for example, per VM lists), do not have pointers to objects in other spaces belonging to a different VM, and thus cannot get access to objects outside of its name space. Through this *contextualization* the global identifiers become per-VM global identifiers. Filters, on the other hand, control access to kernel objects with runtime checks to determine whether the VM has the appropriate permissions. For a hypervisor security isolation is also achieved with contextualization and filters, but generally these apply to constructs at the hardware abstraction layer such as virtual memory address spaces, PCI bus addresses, devices, and privileged instructions.

The techniques used by COS- and hypervisor-based systems for resource isolation are quite similar. Both need to multiplex physical resources such as CPU cycles, i/o bandwidth, and memory/disk storage. The latest generation of the Xen hypervisor architecture focuses on multiplexing the CPU. Control over all other physical resources is delegated to one or more privileged host VMs, which multiplex the hardware on behalf of the guest VMs. Interestingly, when Xen's host VM is based on Linux, the resource controllers used to manage network and disk i/o bandwidth among guest VMs are **identical** to those used by VServer. The two systems simply differ in how they map VMs to these resource controllers.

As a point of reference, the Xen hypervisor for the i32 architecture is about 80K lines of code, the paravirtualized variants of Linux require an additional 15K of device drivers, and a few isolated changes to the core Linux kernel code. In contrast, VServer adds less than 8700 lines of code to the Linux kernel, and due to its mostly architecture independent nature it has been validated to work on eight different instruction set architectures. While lightweight in terms of lines of code involved, VServer introduces 50+ new kernel files and touches 300+ existing ones—representing a nontrivial software-engineering task.

3.2 VServer Resource Isolation

This section describes in what way VServer implements resource isolation. It is mostly an exercise of leveraging existing resource management and accounting facilities already present in Linux. For both physical and logical resources, VServer simply imposes limits on how much of a resource a VM can consume.

3.2.1 CPU Scheduling: Fair Share and Reservations

VServer implements CPU isolation by overlaying a token bucket filter (TBF) on top of the standard $O(1)$ Linux CPU scheduler. Each VM has a token bucket that accumulates tokens at a specified rate; every timer tick, the VM that owns the running process is charged one token. A VM that runs out of tokens has its processes removed from the runqueue until its bucket accumulates a minimum amount of tokens. Originally the VServer TBF was used to put an upper bound on the amount of CPU that any one VM could receive. However, it is possible to express a range of isolation policies with this simple mechanism. We have modified the TBF to provide fair sharing **and/or** work-conserving CPU reservations.

The rate that tokens accumulate in a VM's bucket depends on whether the VM has a *reservation* and/or a *share*. A VM with a reservation accumulates tokens at its reserved rate: for example, a VM with a 10% reservation gets 100 tokens per second, since a token entitles it to run a process for one millisecond. A VM with a share that has runnable processes will be scheduled before the idle task is scheduled, and only when all VMs with reservations have been honored. The end result is that the CPU capacity is effectively partitioned between the two classes of VMs: VMs with reservations get what they've reserved, and VMs with shares split the unreserved capacity of the machine proportionally. Of course, a VM can have both a reservation (e.g., 10%) and a fair share (e.g., 1/10 of idle capacity).

3.2.2 I/O QoS: Fair Share and Reservations

The Hierarchical Token Bucket (HTB) queuing discipline of the Linux Traffic Control facility (tc) [10] is used to provide network bandwidth reservations and fair service among VServer. For each VM, a token bucket is created with a *reserved rate* and a *share*: the former indicates the amount of outgoing bandwidth dedicated to that VM, and the latter governs how the VM shares bandwidth beyond its reservation. Packets sent by a VServer are tagged with its context id in the kernel, and subsequently classified to the VServer's token bucket. The HTB queuing discipline then allows each VServer to send packets at the reserved rate of its token bucket, and fairly distributes the excess capacity to the VServer in proportion to their shares. Therefore, a VM can be given a capped reservation (by specifying a reservation but no share), "fair best effort" service (by specifying a share with no reservation), or a work-conserving reservation (by specifying both).

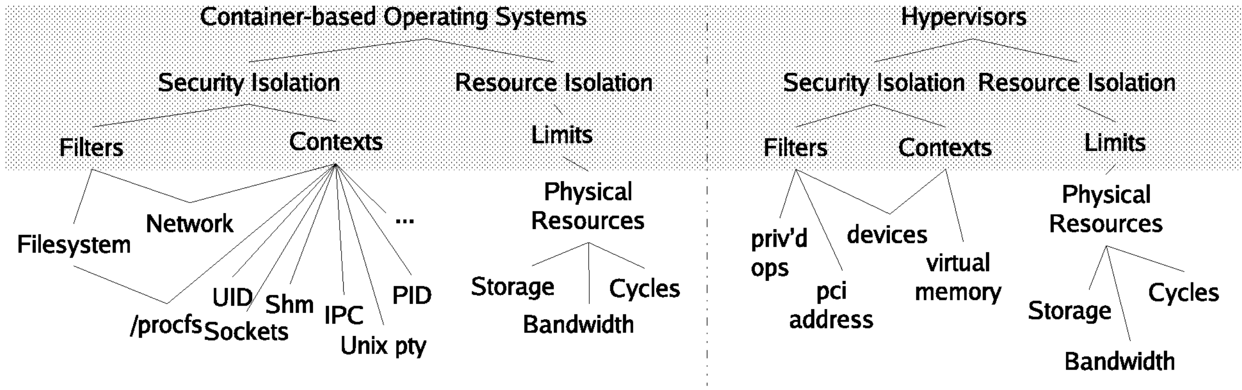


Figure 4: Isolation Taxonomy of COS and Hypervisor-based Systems

Disk I/O is managed in VServer using the standard Linux CFQ ("completely fair queuing") I/O scheduler. The CFQ scheduler attempts to divide the bandwidth of each block device fairly among the VMs performing I/O to that device.

3.2.3 Storage Limits

VServer provides the ability to associate limits to the amount of memory and disk storage a VM can acquire. For disk storage one can specify limits on the max number of disk blocks and inodes a VM can allocate. For memory storage one can specify the following limits: a) the maximum resident set size (RSS), b) number of anonymous memory pages have (ANON), and c) number of pages that may be pinned into memory using mlock() and mlockall() that processes may have within a VM (MEMLOCK). Also, one can declare the number of pages a VM may declare as SYSV shared memory.

Note that fixed upper bounds on RSS are not appropriate for usage scenarios where administrators wish to overbook VMs. In this case, one option is to let VMs compete for memory, and use a watchdog daemon to recover from overload cases—for example by killing the VM using the most physical memory. PlanetLab [17] is one example where memory is a particularly scarce resource, and memory limits without overbooking are impractical: given that there are up to 90 active VMs on a PlanetLab server, this would imply a tiny 10MB allocation for each VM on the typical PlanetLab server with 1GB of memory. Instead, PlanetLab provides basic memory isolation between VMs by running a simple watchdog daemon, called pl_mom, that resets the VM consuming the most physical memory when swap has almost filled. This penalizes the memory hog while keeping the system running for everyone else, and is effective for the workloads that PlanetLab supports. A similar technique is apparently used by managed web hosting companies.

3.3 VServer Security Isolation

VServer makes a number of kernel modifications to enforce security isolation.

3.3.1 Process Filtering

VServer reuses the global PID space across all VMs. In contrast, other container-based systems such as OpenVZ contextualize the PID space per VM. There are obvious benefits to the latter, specifically it eases the implementation of VM checkpoint, resume, and migration more easily as processes can be re-instantiated with the same PID they had at the time of checkpoint. VServer will move to this model, but for the sake of accuracy and completeness we will describe its current model.

VServer filters processes in order to hide all processes outside a VM's scope, and prohibits any unwanted interaction between a process inside a VM and a process belonging to another VM. This separation requires the extension of some existing kernel data structures in order for them to: a) become aware to which VM they belong, and b) differentiate between identical UIDs used by different VMs.

To work around false assumptions made by some user-space tools (like pstree) that the 'init' process has to exist and have PID 1, VServer also provides a per VM mapping from an arbitrary PID to a fake init process with PID 1.

When a VServer-based system boots, all processes belong to a *default host VM*. To simplify system administration, this host VM is no different than any other guest VM in that one can only observe and manipulate processes belonging to that VM. However, to allow for a global process view, VServer defines a special *spectator* VM that can peek at all processes at once.

A side effect of this approach is that process migration from one VM to another VM *on the same host* is achieved by changing its VM association and updating the corresponding per-VM resource usage statistics such NPROC, NOFILE, RSS, ANON, MEMLOCK, etc..

3.3.2 Network Separation

Currently, VServer does not fully virtualize the networking subsystem, as is done by OpenVZ and other container-based systems. Rather, it shares the networking subsystem (route tables, IP tables, etc.) between all VMs, but only lets VMs bind sockets to a set of available IP addresses specified either at VM creation or dynamically by the default host VM. This has the drawback that it does not let VMs change their

route table entries or IP tables rules. However, it was a deliberate design decision to achieve native Linux networking performance at GigE+ line rates.

For VServer's *network separate* approach several issues have to be considered; for example, the fact that bindings to special addresses like IPADDR_ANY or the local host address have to be handled to avoid having one VM receive or snoop traffic belonging to another VM. The approach to get this right involves tagging packets with the appropriate VM identifier and incorporating the appropriate filters in the networking stack to ensure only the right VM can receive them. As will be shown later, the overhead of this is minimal as high-speed networking performance is indistinguishable between a native Linux system and one enhanced with VServer.

3.3.3 The Chroot Barrier

One major problem of the chroot() system used in Linux lies within the fact that this information is volatile, and will be changed on the 'next' chroot() system call. One simple method to escape from a chroot-ed environment is as follows:

- Create or open a file and retain the file-descriptor, then chroot into a subdirectory at equal or lower level with regards to the file. This causes the 'root' to be moved 'down' in the filesystem.
- Use fchdir() on the file descriptor to escape from that 'new' root. This will consequently escape from the 'old' root as well, as this was lost in the last chroot() system call.

VServer uses a special file attribute, known as the Chroot Barrier, on the parent directory of each VM to prevent unauthorized modification and escape from the chroot confinement.

3.3.4 Upper Bound for Linux Capabilities

Because the current Linux Capability system does not implement the filesystem related portions of POSIX Capabilities that would make setuid and setgid executables secure, and because it is much safer to have a secure upper bound for all processes within a context, an additional per-VM capability mask has been added to limit all processes belonging to that context to this mask. The meaning of the individual caps of the capability bound mask is exactly the same as with the permitted capability set.

3.4 VServer Filesystem Unification

One central objective of VServer is to reduce the overall resource usage wherever possible. VServer implements a simple disk space saving technique by using a simple unification technique applied at the whole file level. The basic approach is that files common to more than one VM, which are rarely going to change (e.g., like libraries and binaries from similar OS distributions), can be hard linked on a shared filesystem. This is possible because the guest VMs can safely share filesystem objects (inodes). The technique reduces the amount of disk space, inode caches, and even memory mappings for shared libraries.

The only drawback is that without additional measures, a VM could (un)intentionally destroy or modify such shared files, which in turn would harm/interfere other VMs. The approach taken by VServer is to mark the files as copy-on-write. When a VM attempts to mutate a hard linked file with CoW attribute set, VServer will give the VM a private copy of the file.

Such CoW hard linked files belonging to more than one context are called 'unified' and the process of finding common files and preparing them in this way is called Unification. The reason for doing this is reduced resource consumption, not simplified administration. While a typical Linux Server install will consume about 500MB of disk space, 10 unified servers will only need about 700MB and as a bonus use less memory for caching.

4. SYSTEM EFFICIENCY

This section explores the performance and scalability of COS- and hypervisor-based virtualization. We refer to the combination of performance and scale as the *efficiency* of the system, since these metrics correspond directly to how well the virtualizing system orchestrates the available physical resources for a given workload.

For all tests, VServer performance is comparable to an unvirtualized Linux kernel. Yet, the comparison shows that although Xen3 continues to include new features and optimizations, the overhead required by the virtual memory sub-system still introduces an overhead of up to 49% for shell execution. In terms of absolute performance on server-type workloads, Xen3 lags an unvirtualized system by up to 40% for network throughput while demanding a greater CPU load and 50% longer for disk intensive workloads.

4.1 Configuration

All experiments are run on an HP Proliant DL360 G4p with dual 3.2 GHz Xeon processor, 4GB RAM, two Broadcom NetXtreme GigE Ethernet controllers, and two 160GB 7.2k RPM SATA-100 disks. The Xeon processors each have a 2MB L2 cache. Due to reports [21] indicating that hyper-threading degrades performance for certain environments, we run all tests with hyper-threading disabled. The three kernels under test were compiled for uniprocessor as well as SMP architectures, and unless otherwise noted, all experiments are run within a single VM provisioned with all available resources. In the case of Xen, neither the guest VM nor the hypervisor include device drivers. Instead, a privileged, host VM runs the physical devices and exposes virtual devices to guests. In our tests, the host VM reserves 512MB and the remaining available is available to guest VMs.

The Linux kernel and its variants for Xen and VServer have hundreds of system configuration options, each of which can potentially impact system behavior. We have taken the necessary steps to normalize the effect of as many configuration options as possible, by preserving homogeneous setup across systems, starting with the hardware, kernel configuration, filesystem partitioning, and networking settings. The goal is to ensure that observed differences in performance are a consequence of the virtualization architectures evaluated, rather than a particular set of configuration parameters. Appendix A describes the specific configurations we have used

in further detail.

The hypervisor configuration is based on Xen 3.0.4, which at the time of this writing was the latest stable version available. The corresponding patch to Linux is applied against a 2.6.16.33 kernel. Prior to 3.0.4, we needed to build separate host VM and guest VM kernels, but we now use the unified, Xen paravirtualized Linux kernel that is re-purposed at runtime to serve either as a host or guest VM. Finally, we also build SMP enabled kernels, as this is now supported by the stable Xen hypervisor.

The COS configuration consists of the VServer 2.0.3-rc1 patch applied to the Linux 2.6.16.33 kernel. Our VServer kernel includes several additions that have come as a result of VServer's integration with Planetlab. As discussed earlier, this includes the new CPU scheduler that preserves the existing $O(1)$ scheduler and enables CPU reservations for VMs, and shims that let VServer directly leverage the existing CFQ scheduler to manage disk I/O and the HTB filter to manage network I/O.

4.2 Micro-Benchmarks

While micro-benchmarks are incomplete indicators of system behavior for real workloads [5], they do offer an opportunity to observe the fine-grained impact that different virtualization techniques have on primitive OS operations. In particular, the *OS* subset of McVoy's *lmbench* benchmark [13] version 3.0-a3 includes experiments designed to target exactly these subsystems.

For all three systems, the majority of the tests perform worse in the SMP kernel than the UP kernel. While the specific magnitudes may be novel, the trend is not surprising, since the overhead inherent to synchronization, internal communication, and caching effects of SMP systems is well known. For brevity, the following discussion focuses on the overhead of virtualization using a uniprocessor kernel. While the *lmbench* suite includes a large number of latency benchmarks, Table 2 shows results only for those which represent approximately 2x or greater discrepancy between VServer-UP and Xen3-UP.

For the uniprocessor systems, our findings are consistent with the original report of Barham et al [3] that Xen incurs a penalty for virtualizing the virtual memory hardware. In fact, the pronounced overhead observed in Xen comes entirely from the hypercalls needed to update the guest's page table. This is one of the most common operations in a multiuser system. While Xen3 has optimized page table updates

Configuration	Linux-UP	VServer-UP	Xen3-UP
fork process	86.50	86.90	**271.90**
exec process	299.80	302.00	**734.70**
sh process	968.10	977.70	**1893.30**
ctx (16p/64K)	3.38	3.81	**6.02**
mmap (64MB)	377.00	379.00	**1234.60**
mmap (256MB)	1491.70	1498.00	**4847.30**
page fault	1.03	1.03	**3.21**

Table 2: LMbench OS benchmark timings for uniprocessor kernels – times in μs

relative to Xen2, common operations such as process executing, context switches and page faults still incur observable overhead.

The first three rows in Table 2 show the performance of *fork process*, *exec process*, and *sh process* across the systems. The performance of VServer-UP is always within 1% of Linux-UP. Also of note, Xen3-UP performance has improved over that of Xen2-UP due to optimizations in the page table update code that batch pending transactions for a single call to the hypervisor. Yet, the inherent overhead is still measurable, and almost double in the case of *sh process*.

The next row shows context switch overhead between different numbers of processes with different working set sizes. As explained by Barham [3], the $2\mu s$ to $3\mu s$ overhead for these micro-benchmarks are due to hypercalls from the guest VM into the hypervisor to change the page table base. In contrast, there is little overhead seen in VServer-UP relative to Linux-UP.

The next two rows show *mmap* latencies for 64MB and 256MB files. The latencies clearly scale with respect to to the size of the file, indicating that the Xen kernels incur a $19\mu s$ overhead per megabyte, versus $5.9\mu s$ in the other systems. This is particularly relevant for servers or applications that use *mmap* as a buffering technique or to access large data sets, such as [16].

4.3 System Benchmarks

Two factors contribute to performance overhead in the Xen3 hypervisor system: overhead in network I/O and overhead in disk I/O. Exploring these dimensions in isolation provides insight into the sources of overhead for server workloads in these environments. To do so, we repeat various benchmarks used in the original and subsequent performance measurements of Xen [3, 4, 6]. In particular, there are various multithreaded applications designed to create real-world, multicomponent stresses on a system, such as Iperf, OSDB-IR, and a kernel compile. In addition, we explore several single-threaded applications such as a dd, Dbench and Postmark to gain further insight into the overhead of Xen. These exercise the whole system with a range of server-type workloads illustrating the absolute performance offered by Linux, VServer, and Xen3.

For these benchmarks there is only one guest VM active, and this guest is provisioned with all available memory. For Xen, the host VM is provisioned with 512MB of RAM and a fair share of the CPU. Each reported score is the average of 3 to 5 trials. All results are normalized relative to Linux-UP, unless otherwise stated.

4.3.1 Network Bandwidth Benchmark

Iperf is an established tool [1] for measuring link throughput with TCP or UDP traffic. We use it to measure TCP bandwidth between a pair of systems. We measure both raw throughput and the CPU utilization observed on the receiver. This is done in two separate experiments to avoid measurement overhead interfering with throughput—i.e., max throughput is lower when recording CPU utilization with *sysstat* package on VServer and Linux, and *XenMon* on Xen.

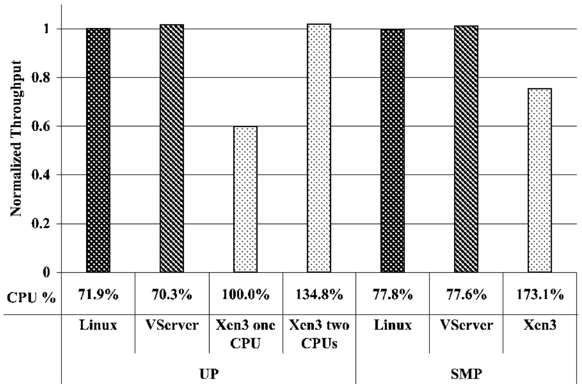


Figure 5: Iperf TCP bandwidth and CPU utilization.

Figure 5 illustrates both the throughput achieved and the aggregate percentage of CPU necessary to achieve this rate on the receiver. The first three columns are trials run with the uniprocessor kernels. Both Linux and VServer on a single processor achieve line rate with just over 70% CPU utilization as the data sink. In contrast, the Xen3-UP configuration can only achieve 60% of the line rate, because having the host VM, guest VM, and hypervisor all pinned to a single CPU saturate the CPU due to the overhead of switching between VMs and interaction with the hypervisor.

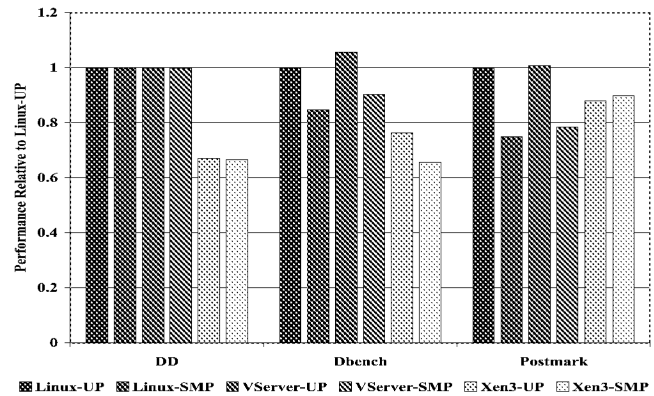
The fourth column labeled 'Xen3 two CPUs' consists of the same Xen3-UP configuration, except the host and guest VMs are pinned to separate CPUs. In this way the host and guest VMs do not interfere with each other, and can achieve line rate just as Linux and VServer. This is an improvement over prior versions of Xen using the same hardware. We attribute the improvement to switching from a safe, page-flipping data transfer model to one utilizing memory copies between VMs. Still, when compared to Linux and VServer running on a single CPU, the overall CPU utilization of the Xen3-UP configuration running on two CPUs is nearly 2x the load experienced by Linux or VServer.

The last three columns use SMP variants of the Linux kernels. Again VServer compares closely to Linux. Interestingly, Xen3 with a SMP guest VM cannot achieve line rate. It saturates the CPU it shares with the host VM. And as a result it also performs worse compared to the Linux-UP configuration.

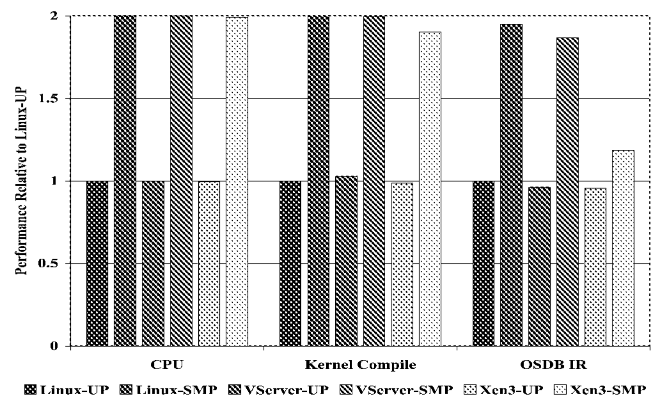
4.3.2 *Macro Benchmarks*

This section evaluates a number of benchmarks that are CPU and/or disk I/O intensive. The results of these benchmarks are shown in Figures 6(a) and 6(b), which summarizes the performance between VServer and Xen3 normalized against Linux.

The DD benchmark writes a 6GB file to a scratch device. Linux and VServer have identical performance for this benchmark, as the code path for both is basically identical. In contrast, for Xen3 we observe significant slow down for both UP and SMP. This is due to additional buffering, copying, and synchronization between the host VM and guest VM to write blocks to disk.



(a) Disk performance



(b) Performance of CPU and memory bound benchmarks

Figure 6: Relative performance of Linux, VServer, and XenU kernels.

DBench is derived from the industry-standard NetBench filesystem benchmark and emulates the load placed on a file server by Windows 95 clients. The DBench score represents the throughput experienced by a single client performing around 90,000 file system operations. Because DBench is a single-threaded application, the Linux-SMP and VServer-SMP results show reduced performance due to inherent overhead of SMP systems. Accordingly, the Xen3-UP performance is modestly greater than that of Xen3-SMP, but again, both have performance that is 25-35% less than Linux-UP, while VServer-UP slightly exceeds the Linux performance.

Postmark [8] is also a single-threaded benchmark originally designed to stress filesystems with many small file operations. It allows a configurable number of files and directories to be created, followed by a number of random transactions on these files. In particular, our configuration specifies 100,000 files and 200,000 transactions. Postmark generates many small transactions like those experienced by a heavily loaded email or news server, from which it derives the name 'postmark'. Again, the throughput of Xen3 is less

than both Linux and VServer, as there is more overhead involved in pushing filesystem updates from the guest VM via the host VM to the disk device.

Figure 6(b) demonstrates the relative performance of several CPU and memory bound activities. These tests are designed to explicitly avoid the I/O overhead seen above. Instead, inefficiency here is a result of virtual memory, scheduling or other intrinsic performance limits. The first test is a single-threaded, CPU-only process. When no other operation competes for CPU time, this process receives all available system time. But, the working set size of this process fits in processor cache, and does not reveal the additive effects of a larger working set, as do the second and third tests.

The second test is a standard kernel compile. It uses multiple threads and is both CPU intensive as well as exercising the filesystem with many small file reads and creates. However, before measuring the compilation time, all source files are moved to a RAMFS to remove the impact of any disk effects. The figure indicates that performance is generally good for Xen relative to Linux-UP, leaving overheads only in the range of 1% for Xen3-UP to 7% for Xen3-SMP.

Finally, the Open Source Database Benchmark (OSDB) provides realistic load on a database server from multiple clients. We report the Information Retrieval (IR) portion, which consists of many small transactions, all reading information from a 40MB database, again cached in main memory. The behavior of this benchmark is consistent with current web applications. Again, performance of VServer-UP is comparable to that of Linux-UP within 4%, but Xen3-SMP suffers a 39% overhead relative to Linux-SMP. Not until we look at the performance of this system at scale do the dynamics at play become clear.

4.4 Performance at Scale

This section evaluates how effectively the virtualizing systems provide performance at scale. Barham et al. point out that unmodified Linux cannot run multiple instances of PostgreSQL due to conflicts in the SysV IPC namespace. However, VServer's mechanisms for security isolation contain the SysV IPC namespace within each context. Therefore, using OSDB, we simultaneously demonstrate the security isolation available in VServer that is unavailable in Linux, and the superior performance available at scale in a COS-based design.

The Information Retrieval (IR) component of the OSDB package requires memory and CPU time. If CPU time or system memory is dominated by any one VM, then the others will not receive a comparable share, causing aggregate throughput to suffer. As well, overall performance is calculated as a function of the finish-time of all tests. This methodology favors schedulers which are able to keep the progress of each VM in sync, such that all tests end simultaneously. Figure 7 shows the results of running 1, 2, 4, and 8 simultaneous instances of the OSDB IR benchmark. Each VM runs an instance of PostgreSQL to serve the OSDB test.

A virtualization solution with strong isolation guarantees would partition the share of CPU time, buffer cache, and memory bandwidth perfectly among all active VMs and

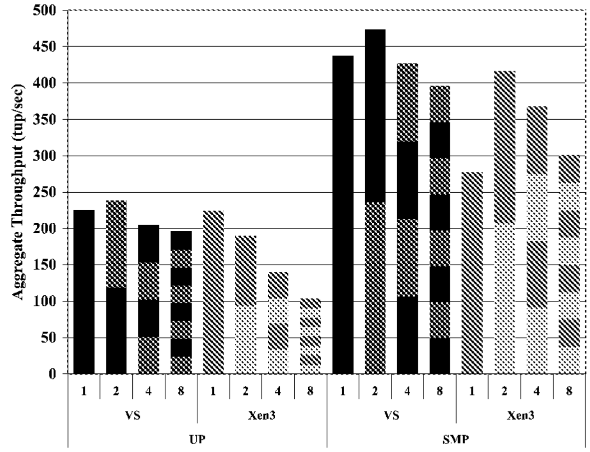


Figure 7: OSDB-IR at Scale. Performance across multiple VMs

maintain the same aggregate throughput as the number of active VMs increased. However, for each additional VM, there is a linear increase in the number of processes and the number of I/O requests. Since it is difficult to perfectly isolate all performance effects, the intensity of the workload adds increasingly more pressure to the system and eventually, aggregate throughput diminishes. Figure 7 illustrates that after an initial boost in aggregate throughput at two VMs, all systems follow the expected diminishing trend.

We observe two noteworthy deviations from this trend. First, Xen3-UP does not improve aggregate throughput at two VMs. Instead, the Xen3-UP performance quickly degrades for each test. In part this is due to the increased load on the single CPU. It must simultaneously host as many as 8 guest VMs as well as the host VM. Because the host VM is explicitly scheduled just as the guest VMs, for larger tests it is given more hosts to serve and correspondingly less time is available for each guest. This pressure is worst when the load reaches eight simultaneous VMs, where the performance is 47% less than VServer-UP.

Second, the significant performance jump between one and two VMs for Xen3-SMP is very large. This behavior is due in part to the credit scheduler unequally balancing the Xen3-SMP kernel across both physical CPUs, as evidenced by monitoring the CPU load on both processors. As a result, the single Xen3-SMP case does not have the opportunity to benefit from the full parallelism available. Not until this situation is duplicated with two Xen3-SMP kernels is greater utility of the system achieved. Of course, VServer-SMP outperforms the Xen3-SMP system. In particular, the total performance in the VServer, eight VM case is within 5% of Xen3-SMP with 2 VMs and greater than any of the other Xen3-SMP tests.

Two factors contribute to the the higher average performance of VServer: lower overhead imposed by the COS approach and a better CPU scheduler for keeping competing VMs progressing at the same rate. As a result, there is simply more CPU time left to serve clients at increasing scale.

Fraction of Host Requested	VS-UP Achieved	VS-SMP Achieved	Xen3-UP Achieved	Xen3-SMP Achieved
Weight $1/4^{th}$	25.16%	49.88%	**15.51%**	**44.10%**
Cap $1/4^{th}$	n/a	n/a	24.35%	**46.62%**

Table 3: Percent of time achieved from a one quarter CPU reservations using weights and caps. Deviations are highlighted.

5. ISOLATION

VServer complements Linux's existing per process resource limits with per VM limits for logical resources such as shared file descriptors, number of process limits, shared memory sizes, etc., which are shared across the whole system. In this way VServer can effectively isolate VMs running fork-bombs and other antisocial activity through the relevant memory caps, process number caps, and other combinations of resource limits. In contrast, Xen primarily focuses on CPU scheduling and memory limits, while i/o bandwidth management is left to the host VM. As mentioned, for both Xen and VServer the disk and network i/o bandwidth management mechanism (CFQ and HTB, respectively) are largely identical—differing only in the shims that map per VM activities into CFQ and HTB queues. What differs significantly between Xen and VServer are their CPU schedulers. The remainder of this section first focuses on how well their schedulers can fairly share and reserve the CPU among VMs, and then evaluates at a macro level the impact of introducing an antisocial process contained in one VM on another VM running the OSDB benchmark.

5.1 CPU Fair Share and Reservations

To investigate both isolation of a single resource and resource guarantees, we use a combination of CPU intensive tasks. Hourglass is a synthetic real-time application useful for investigating scheduling behavior at microsecond granularity [20]. It is CPU-bound and involves no I/O.

Eight VMs are run simultaneously. Each VM runs an instance of hourglass, which records contiguous periods of time scheduled. Because hourglass uses no I/O, we may infer from the gaps in its time-line that either another VM is running or the virtualized system is running on behalf of another VM, in a context switch for instance. The aggregate CPU time recorded by all tests is within 1% of system capacity.

We evaluated two experiments: 1) all VMs are given the same fair share of CPU time, and 2) one of the VMs is given a reservation of $1/4^{th}$ of overall CPU time. For the first experiment, VServer and Xen for both UP and SMP systems do a good job at scheduling the CPU among the VMs such that each receive approximately one eights of the available time.

Table 3 reports the amount of CPU time received when the CPU reservation of one VM is set to one fourth of the system. For a two-way SMP system with an aggregate of 200% CPU time, a one fourth reservation corresponds to 50% of available resources. The CPU scheduler for VServer achieves this within 1% of the requested reservation for both UP and SMP configurations.

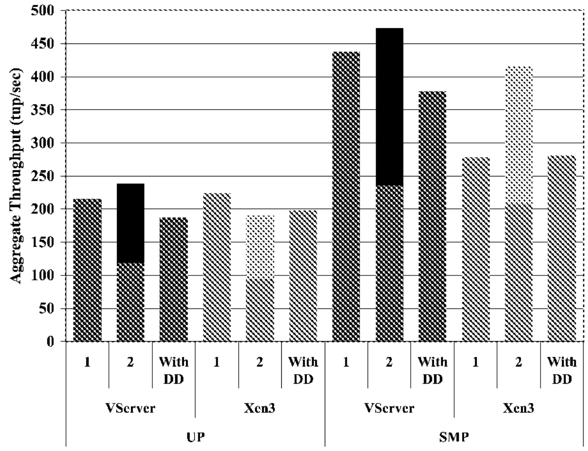


Figure 8: Database performance with competing VMs

In contrast, Xen is off by up to 6% of the requested reservations in the worst case. The reason for this is because Xen does not offer an explicit means to specify a reservation of CPU time. Instead, it provides only two CPU time allocation parameters: relative weights or performance caps (i.e., hard limits). The inherent problem with relative weights is that there is no 1:1 relationship between a VM's weight and the minimum percentage of CPU allotted to it under load. Performance caps, on the other hand, only let one express the maximum fraction of the system a VM can consume, but not the minimum it should obtain. As a result, to express a reservation in terms of weights or performance caps can at best be approximated, which the results shown in Table 3 demonstrate.

5.2 Performance Isolation

Traditional time-sharing UNIX systems have a legacy of vulnerability to layer-below attacks, due to unaccounted, kernel resource consumption. To investigate whether VServer is still susceptible to such interference, we elected to perform a variation of the multi-OSDB database benchmark. Now, instead of all VMs running a database, one will behave *maliciously* by performing a continuous *dd* of a 6GB file to a separate partition of a disk common to both VMs.

Figure 8 shows that the performance of OSDB on VServer is impacted between 13-14% for both UP and SMP when competing with an active *dd*. This, despite the fact that the VServer block cache is both global (shared by all VMs) and not directly accounted to the originating VM. Earlier kernels, such as 2.6.12 kernel, experienced crippling performance penalties during this test while the swap file was enabled. While the ultimate cause of this overhead in older kernels is still not known, these results for modern kernels are very a promising improvement. Clearly, though, additional improvements can be made to account more completely for guest block cache usage and are a subject for future research.

Xen, on the other hand, explicitly partitions the physical memory to each VM. As a result, since the block cache is maintained by each kernel instance, the guests are not vul-

nerable to this attack. Yet, Xen-UP sees a 12% decrease for and Xen3-SMP actually gets a 1% boost. Given earlier results these are not surprising. Any additional activity by the *dd* VM or the host VM acting on its behalf, is expected to take processing time away from the Xen3-UP, OSDB VM. As for the Xen3-SMP case, the original poor performance of the single VM was due to poor scheduling by the credit scheduler, which allowed an uneven balancing across the physical CPUs. Given this scenario, the lighter loaded CPU runs the *dd* VM, which requires little CPU time and consequently has little impact on the mostly OSDB VM.

6. CONCLUSION

Virtualization technology benefits a wide variety of usage scenarios. It promises such features as configuration independence, software interoperability, better overall system utilization, and resource guarantees. This paper has compared two modern approaches to providing these features while they balance the tension between complete isolation of co-located VMs and efficient sharing of the physical infrastructure on which the VMs are hosted.

We have shown the two approaches share traits in their high-level organization. But some features are unique to the platform. Xen is able to support multiple kernels while by design VServer cannot. Xen also has greater support for virtualizing the network stack and allows for the possibility of VM migration, a feature that is possible for a COS design, but not yet available in VServer. VServer, in turn, maintains a small kernel footprint and performs equally with native Linux kernels in most cases.

Unfortunately, there is no one-size solution. As our tests have shown, i/o related benchmarks perform worse on Xen when compared to VServer. This is an artifact of virtualizing i/o devices via a proxy host VM. VMware partially addresses this issue by incorporating device drivers for a small set of supported high performance I/O devices directly into its ESX hypervisor product line. In contrast, this issue is non-issue for COS based systems where all I/O operates at native speeds. Despite these weaknesses we expect ongoing efforts to continue to improve Xen-based hypervisor solutions.

In the mean time, for managed web hosting, PlanetLab, etc., the trade-off between isolation and efficiency is of paramount importance. Our experiments indicate that container-based systems provide up to 2x the performance of hypervisor-based systems for server-type workloads and scale further while preserving performance. And, we expect that container-based systems like VServer will incorporate more of the feature set that draws users to hypervisors (e.g., full network virtualization, migration, etc.), and thereby continue to compete strongly against hypervisor systems like Xen for these usage scenarios.

7. REFERENCES

[1] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. Iperf version 1.7.1. `http://dast.nlanr.net/Projects/Iperf/`.

[2] G. Banga, P. Druschel, and J. C. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Proc. 3rd OSDI*, pages 45–58, New Orleans, LA, Feb 1999.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. 19th SOSP*, Lake George, NY, Oct 2003.

[4] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J. Matthews. Xen and the art of repeated research. In *USENIX Technical Conference FREENIX Track*, June 2004.

[5] R. P. Draves, B. N. Bershad, and A. F. Forin. Using Microbenchmarks to Evaluate System Performance. In *Proc. 3rd Workshop on Workstation Operating Systems*, pages 154–159, Apr 1992.

[6] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. W. eld, and M. Williamson. Safe Hardware Access with the Xen Virtual Machine Monitor. In *First Workshop on Operating System and Architectural Support for the On-Demand IT Infrastructure (OASIS)*, Oct 2004.

[7] P.-H. Kamp and R. N. M. Watson. Jails: Confining the Omnipotent Root. In *Proc. 2nd Int. SANE Conf.*, Maastricht, The Netherlands, May 2000.

[8] J. Katcher. Postmark: a new file system benchmark. In *TR3022. Network Appliance*, October 1997.

[9] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. T. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE J. Sel. Areas Comm.*, 14(7):1280–1297, 1996.

[10] Linux Advanced Routing and Traffic Control. `http://lartc.org/`.

[11] Linux-VServer Project. `http://linux-vserver.org/`.

[12] B. McCarty. *SELINUX: NSA's open source Security Enhanced Linux*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, USA, 2005.

[13] L. McVoy and C. Staelin. lmbench: Portable Tools for Performance Analysis. In *Proc. USENIX '96*, pages 279–294, Jan 1996.

[14] S. Nabah, H. Franke, J. Choi, C. Seetharaman, S. Kaplan, N. Singhi, V. Kashyap, and M. Kravetz. Class-based prioritized resource control in Linux. In *Proc. OLS 2003*, Ottawa, Ontario, Canada, Jul 2003.

[15] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proc. 5th OSDI*, pages 361–376, Boston, MA, Dec 2002.

[16] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable Web server. In *Proceedings of the USENIX 1999 Annual Technical Conference*, 1999.

[17] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. Experiences building planetlab. In *Proceedings of the 7th USENIX Symposium on Operating System Design and Implementation (OSDI '06)*, Seattle, WA, November 2006.

[18] S. Potter and J. Nieh. Autopod: Unscheduled system updates with zero data loss. In *Abstract in Proceedings of the Second IEEE International Conference on Autonomic Computing (ICAC 2005)*, June 2005.

[19] D. Price and A. Tucker. Solaris zones: Operating

system support for consolidating commercial workloads. In *Proceedings of the 18th Usenix LISA Conference.*, 2004.

[20] J. Regehr. Inferring scheduling behavior with hourglass. In *In Proceedings of the Freenix Track of the 2002 USENIX Annual Technical Conference*, June 2002.

[21] Y. Ruan, V. S. Pai, E. Nahum, and J. M. Tracey. Evaluating the impact of simultaneous multithreading on network servers using real hardware. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 315–326, New York, NY, USA, 2005. ACM Press.

[22] M. M. Swift, B. N. Bershad, and H. M. Levy. Improving the reliability of commodity operating systems. *ACM Trans. Comput. Syst.*, 23(1):77–110, 2005.

[23] SWSoft. Virtuozzo Linux Virtualization. `http://www.virtuozzo.com`.

[24] Vivek Pai and KyoungSoo Park. CoMon: A Monitoring Infrastructure for PlanetLab. `http://comon.cs.princeton.edu`.

[25] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, Aug 2002.

APPENDIX

A. NORMALIZED CONFIGURATION

A significant aspect of this work involved ensuring that the experiments were fair. We report the configuration details of the various subsystems.

A.1 Hardware

All experiments are run on an HP DL360 Proliant with dual 3.2 GHz Xeon processor, 4GB RAM, two Broadcom NetXtreme GigE Ethernet controllers, and two 160GB 7.2k RPM SATA-100 disks. The Xeon processors each have a 2MB L2 cache. All tests are run with hyper-threading disabled.

A.2 Kernel Configuration

All kernels were based on the 2.6.16.33 Linux kernel. The kernel configuration options were normalized across all platforms. The only differences between the kernel config files that remain come from specific options available for the given platform, i.e. VServer or Xen specific, for which there is no comparable option available in the other versions.

A.3 System Clock

The Linux, VServer and unified XenoLinux kernels are configured to run with a 250Hz system clock. This is a deviation from the default configuration of both VServer and Xen, whose defaults are 1000Hz and 100Hz respectively. However, the ten fold difference between the two was in earlier tests shown to contribute to latency measurements, and context switch overheads. Keeping the system clock equal puts both on equal footing.

A.4 Filesystem

The host VM is a Fedora Core 5 distribution with current updates. This is the environment into which the host VMs boot up. The distribution populating the guest VMs is Fedora Core 2 with the all current updates.

Each guest partition is a 2GB, LVM-backed, ext3 filesystem with the following features: has_journal, filetype, and sparse_super. No other fs level features are enabled. The default journal size is created by mke2fs. Due to the size of our disk and the chosen partition size we are limited to approximately 80 VMs. Sixty-four are currently available. The remaining space is used to host larger scratch space for particular tests, DD for instance.

We account for the natural, diminishing read and write performance across the extent of platter-based hard drives by assigning each virtual machine a dedicated LVM partition. This partition is used exclusively by one VM, irrespective of which virtualizing system is currently active. This configuration departs from a traditional VServer system, where a file-level copy-on-write technique replicates the base environment for additional storage savings.

A.5 Networking

The physical host has two Ethernet ports, so both Linux and VServer share two IPv4 IP addresses across all VMs. As a consequence, the port space on each IP address is also shared between all VMs. The Xen configuration, on the other hand, differs by virtue of running an autonomous kernel in each VM which includes a dedicated TCP/IP stack and IP address. The Xen network runs in bridged mode for networking tests. We also use two client machines attached to each Ethernet port on the system under test through a 1 Gbps Netgear switch. Each client is a 3.2GHz HP DL320g5 server equipped with a Broadcom Gigabit Ethernet running Fedora Core 5.

To: creganoa@addmg.com,,
From: PAIR_eOfficeAction@uspto.gov
Cc: PAIR_eOfficeAction@uspto.gov
Subject: Private PAIR Correspondence Notification for Customer Number 27975

Dec 10, 2008 05:55:59 AM

Dear PAIR Customer:

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791
UNITED STATES

The following USPTO patent application(s) associated with your Customer Number, 27975 , have new outgoing correspondence. This correspondence is now available for viewing in Private PAIR.

The official date of notification of the outgoing correspondence will be indicated on the form PTOL-90 accompanying the correspondence.

Disclaimer:
The list of documents shown below is provided as a courtesy and is not part of the official file wrapper. The content of the images shown in PAIR is the official record.

Application	Document	Mailroom Date	Attorney Docket No.
10939903	NOA	12/10/2008	78802 (120-1 US)
	NOA	12/10/2008	78802 (120-1 US)
	892	12/10/2008	78802 (120-1 US)

To view your correspondence online or update your email addresses, please visit us anytime at https://sportal.uspto.gov/secure/myportal/privatepair.

If you have any questions, please email the Electronic Business Center (EBC) at EBC@uspto.gov with 'e-Office Action' on the subject line or call 1-866-217-9197 during the following hours:

Monday - Friday 6:00 a.m. to 12:00 a.m.

Thank you for prompt attention to this notice,

UNITED STATES PATENT AND TRADEMARK OFFICE

PATENT APPLICATION INFORMATION RETRIEVAL SYSTEM

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)

EFILED (Signature)

(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

TITLE OF INVENTION: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$755	$300	$0	$1055	03/10/2009

EXAMINER	ART UNIT	CLASS-SUBCLASS
BAUM, RONALD	2439	713-167000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list

(1) the names of up to 3 registered patent attorneys or agents OR, alternatively,

(2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 ALLEN, DYER, DOPPELT ~~MILBRATH & GILCHRIST, P.A.~~
2 Attorneys at Law
255 S. Orange Ave., Ste.1401
3 PO Box 3791
Orlando, FL 32802-3791

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE: Trigence Corp.

(B) RESIDENCE: (CITY and STATE OR COUNTRY): Ottawa, Canada

Please check the appropriate assignee category or categories (will not be printed on the patent) : ☐ Individual ☒ Corporation or other private group entity ☐ Government

4a. The following fee(s) are submitted:
☒ Issue Fee
☒ Publication Fee (No small entity discount permitted)
☐ Advance Order - # of Copies ____________

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**
☐ A check is enclosed.
☒ Payment by credit card. ~~Form PTO-2038 is attached.~~ Paid by ESF Web.
☒ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number 50-2810 (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)
☐ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27.
☐ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature /CHRISTOPHER F. REGAN/ Date 5 March 2009

Typed or printed name Christopher F. Regan Registration No. 34,906

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

Electronic Patent Application Fee Transmittal

Application Number:	10939903
Filing Date:	13-Sep-2004
Title of Invention:	SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS
First Named Inventor/Applicant Name:	Donn Rochette
Filer:	Christopher F. Regan/Lois Love
Attorney Docket Number:	78802 (120-1 US)

Filed as Small Entity

Utility under 35 USC 111(a) Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				
Utility Appl issue fee	2501	1	755	755
Publ. Fee- early, voluntary, or normal	1504	1	300	300

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Extension-of-Time:				
Miscellaneous:				
		Total in USD ($)		**1055**

Electronic Acknowledgement Receipt

EFS ID:	4909815
Application Number:	10939903
International Application Number:	
Confirmation Number:	5216
Title of Invention:	SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS
First Named Inventor/Applicant Name:	Donn Rochette
Customer Number:	27975
Filer:	Christopher F. Regan/Lois Love
Filer Authorized By:	Christopher F. Regan
Attorney Docket Number:	78802 (120-1 US)
Receipt Date:	05-MAR-2009
Filing Date:	13-SEP-2004
Time Stamp:	13:52:20
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$1055
RAM confirmation Number	9296
Deposit Account	502810
Authorized User	

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)

1	Issue Fee Payment (PTO-85B)	78802IF_120_1_US.pdf	129009 7c77b4b03925d90ee049aaeea0074fe3999 4f012	no	1
Warnings:					
Information:					
2	Fee Worksheet (PTO-06)	fee-info.pdf	31730 def4de412c3c38fede54d2e147a44bcacbfa 1e09	no	2
Warnings:					
Information:					
			Total Files Size (in bytes):	160739	

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

<u>New Applications Under 35 U.S.C. 111</u>
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

<u>National Stage of an International Application under 35 U.S.C. 371</u>
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

<u>New International Application Filed with the USPTO as a Receiving Office</u>
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission
I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)
EFILED (Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

TITLE OF INVENTION: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$755	$300	~~$0~~ $1055	$1055	03/10/2009

EXAMINER	ART UNIT	CLASS-SUBCLASS
BAUM, RONALD	2439	713-167000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).
☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list
(1) the names of up to 3 registered patent attorneys or agents OR, alternatively,
(2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

ALLEN, DYER, DOPPELT
1 ~~MILBRATH & GILCHRIST, P.A.~~
2 Attorneys at Law
255 S. Orange Ave., Ste.1401
3 PO Box 3791
Orlando, FL 32802-3791

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE: AppZero Corp.
(B) RESIDENCE: (CITY and STATE OR COUNTRY): Ottawa, Canada

Please check the appropriate assignee category or categories (will not be printed on the patent) : ☐ Individual ☒ Corporation or other private group entity ☐ Government

4a. The following fee(s) are submitted:
☐ Issue Fee
☐ Publication Fee (No small entity discount permitted)
☐ Advance Order - # of Copies ____________

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**
☐ A check is enclosed. Previously Paid 3/5/09
☐ Payment by credit card. Form PTO-2038 is attached.
☒ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number 50-2810 (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)
☐ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27.
☐ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature /CHRISTOPHER F. REGAN/ Date 5 March 2009

Typed or printed name Christopher F. Regan Registration No. 34,906

Electronic Acknowledgement Receipt

EFS ID:	4911186
Application Number:	10939903
International Application Number:	
Confirmation Number:	5216
Title of Invention:	SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS
First Named Inventor/Applicant Name:	Donn Rochette
Customer Number:	27975
Filer:	Christopher F. Regan/Lois Love
Filer Authorized By:	Christopher F. Regan
Attorney Docket Number:	78802 (120-1 US)
Receipt Date:	05-MAR-2009
Filing Date:	13-SEP-2004
Time Stamp:	15:09:07
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Issue Fee Payment (PTO-85B)	Replacement78802IF_120_1_US.pdf	128473 4b6500beddcb4b3b231d74f97605190d19ff52ad	no	1

Warnings:

Information:

Total Files Size (in bytes):	128473

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450**
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

03/05/2009 INTEFSW 00009296 10939903

01 FC:2501 755.00 DA
02 FC:1504 300.00 DA

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)

EFILED (Signature)

(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

TITLE OF INVENTION: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$755	$300	$0	$1055	03/10/2009

EXAMINER	ART UNIT	CLASS-SUBCLASS
BAUM, RONALD	2439	713-167000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list

(1) the names of up to 3 registered patent attorneys or agents OR, alternatively,

(2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 ALLEN, DYER, DOPPELT, ~~MILBRATH & GILCHRIST, P.A.~~
2 Attorneys at Law
3 255 S. Orange Ave., Ste.1401 PO Box 3791 Orlando, FL 32802-3791

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE: Trigence Corp.

(B) RESIDENCE: (CITY and STATE OR COUNTRY): Ottawa, Canada

Please check the appropriate assignee category or categories (will not be printed on the patent): ☐ Individual ☒ Corporation or other private group entity ☐ Government

4a. The following fee(s) are submitted:

☒ Issue Fee

☒ Publication Fee (No small entity discount permitted)

☐ Advance Order - # of Copies ______

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**

☐ A check is enclosed.

☒ Payment by credit card. XXXXXXXXXXXXXXX Paid by ESF Web.

☒ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number 50-2810 (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)

☐ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27.

☐ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature /CHRISTOPHER F. REGAN/ Date 5 March 2009

Typed or printed name Christopher F. Regan Registration No. 34,906

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	ISSUE DATE	PATENT NO.	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	04/14/2009	7519814	78802 (120-1 US)	5216

27975 7590 03/25/2009

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

ISSUE NOTIFICATION

The projected patent number and issue date are specified above.

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment is 933 day(s). Any patent to issue from the above-identified application will include an indication of the adjustment on the front page.

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at (571)-272-4200.

APPLICANT(s) (Please see PAIR WEB site http://pair.uspto.gov for additional applicants):

Donn Rochette, Fenton, IA;
Paul O'Leary, Kanata, CANADA;
Dean Huffman, Kanata, CANADA;

To: creganoa@addmg.com,,
From: PAIR_eOfficeAction@uspto.gov
Cc: PAIR_eOfficeAction@uspto.gov
Subject: Private PAIR Correspondence Notification for Customer Number 27975

Mar 26, 2009 05:42:08 AM

Dear PAIR Customer:

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791
UNITED STATES

The following USPTO patent application(s) associated with your Customer Number, 27975 , have new outgoing correspondence. This correspondence is now available for viewing in Private PAIR.

The official date of notification of the outgoing correspondence will be indicated on the form PTOL-90 accompanying the correspondence.

Disclaimer:
The list of documents shown below is provided as a courtesy and is not part of the official file wrapper. The content of the images shown in PAIR is the official record.

Application	Document	Mailroom Date	Attorney Docket No.
10939903	ISSUE.NTF	03/25/2009	78802 (120-1 US)

To view your correspondence online or update your email addresses, please visit us anytime at https://sportal.uspto.gov/secure/myportal/privatepair.

If you have any questions, please email the Electronic Business Center (EBC) at EBC@uspto.gov with 'e-Office Action' on the subject line or call 1-866-217-9197 during the following hours:

Monday - Friday 6:00 a.m. to 12:00 a.m.

Thank you for prompt attention to this notice,

UNITED STATES PATENT AND TRADEMARK OFFICE

PATENT APPLICATION INFORMATION RETRIEVAL SYSTEM